UML PROFILING AND DSL

version 17.0.1

user guide

No Magic, Inc.

2011

CONTENTS

CONTENTS

PROFILING AND DOMAIN SPECIFIC CUSTOMIZATIONS

Introduction

View Online Demo Domain Specific Language (DSL)

NOTE Domain Specific Language customization engine is available in Standard, Professional, Architect, and Enterprise editions only.

UML is a general purpose visual modeling language for specifying, constructing and documenting the artifacts of systems that can be used with all major application domains and implementation platforms. It has been widely adopted by both industry and academia as the standard language for describing software systems.

However, the fact that UML is a general-purpose notation may limit its suitability for modeling some particular specific domains (e.g. web applications or business processes), for which specialized languages and tools may be more appropriate.

OMG defines two possible approaches for defining domain-specific languages. The first one is based on the definition of a new language, an alternative to UML, using the mechanisms provided by OMG for defining object-based visual languages (i.e., the same mechanisms that have been used for defining UML and its metamodel). In this way, the syntax and semantics of the elements of the new language are defined to fit the domain's specific characteristics. The problem is that standard UML tools will not be able to deal with such a new language.

The second alternative is based on the particularization of UML by specializing some of its elements, imposing new restrictions on them but respecting the UML metamodel, and without modifying the original semantics of the UML elements (i.e., the properties of the UML classes, associations, attributes, etc., will remain the same, but new constraints will be added to their original definitions and relationships).

UML provides a set of extension mechanisms (stereotypes, tag definitions, and constraints) for specializing its elements, allowing customized extensions of UML for particular application domains.

These customizations are sets of UML extensions, grouped into UML profiles. However UML profiles for specific domain can not play role of specialized tool, they are just specialized metamodels.

MagicDraw UML tool provides ability to use brand-new engine for adapting domain specific profiles to create your own custom diagrams, custom specification dialogs, custom real-time semantic rules and other. In other words user is able to create low-budget specialized domain-specific tool and hide UML underneath.

DSL customization is model-driven approach, based on UML profiling.

DEFINITIONS	
Profile	A special kind of package that holds a collection of stereotypes.
Metaclass	A type defined in UML specification (like Class, Component, Package, State etc.). All metaclasses are available in UML Standard Profile::UML2 Metamodel.
Module	A project that has its content shared and used in another project by reference (not contained).

DEFINITIONS	
DSL	Domain Specific Language
DSL Customization	A special profiling for MagicDraw adaptation to some DSL.
DSL type	A stereotype customized to be as new independent type.
Customization class	A class with the «Customization» stereotype applied.
Customization model	A collection of customization classes.
Simply stereotyped element	An element with not customized stereotype applied.
Stereotyped element	An element with customized stereotype applied.
OCL	Object Constraint Language

Advanced customization engines

MagicDraw introduces several advanced customization engines, based on UML Profiles:

- **Custom Diagram Wizard**
  allows creating your own diagram types for custom profile, with your own toolbars, stereotyped elements, symbol styles and custom smart manipulators.
  Such customization is saved in the special “diagram descriptor”, that could be exchanged between users, allowing them use your own custom diagrams.
- **Domain Specific Language Customization Engine (DSL customization engine)** allows “tuning” domain specific profiles, customizing multiple GUI, model initialization, and semantic rules, creating your own Specification dialogs.
  DSL customization is model-driven approach, based on UML profiling. Customization is saved as a UML model.
- **Advanced UML Profiling**
  allows using some profiling enhancements that are not defined in UML, but that helps to solve some common problems like tag grouping, unwanted stereotypes and tags hiding etc.

Below is the list of MagicDraw areas that are possible to customize using MagicDraw advanced customization engines.

	Customization	**Customization engine**
Custom diagrams	Diagram type name	Custom diagram wizard
	Base diagram type	Custom diagram wizard
	Diagram abbreviation	Custom diagram wizard
	Used profiles	Custom diagram wizard
	Diagram toolbar customization	Custom diagram wizard
	Symbol styles	Custom diagram wizard

	Customization	Customization engine
Custom shapes	Scalable icon	Profiling
	Custom draw and layout rules	Open API
	Smart manipulators	Custom diagram wizard
	Context actions for property changing	DSL engine
	Keywords	DSL engine
	Representation text	DSL engine
Custom paths	Line style	Profiling
	Line width	Symbol properties
	Arrow types or end icons	Profiling
Custom model initialization and constraints	Default values for primitive UML properties.	Custom diagram wizard
	Default values for tags with primitive types	Profiling
	Default super classes and interfaces	DSL engine
Connection rules	Allowed metaclasses or stereotypes for end	DSL engine
	Stereotypes that must be applied on relationship ends	DSL engine
Custom Specification dialogs	Grouping into nodes	DSL engine
	Native UML properties on/off	DSL engine
	Always visible custom properties	DSL engine

Exchanging Profiles via XMI

Using UML XMI interchange mechanisms it is possible to interchange profiles between tools, together with models to which they have been applied. A profile must therefore be defined as an interchangeable UML model. In addition to exchanging profiles together with models between tools, profile application should also be definable “by reference”; that is, a profile does not need to be interchanged if it is already present in the importing tool.

For the instructions how to create and use the interchangeable profile, see “Working with UML Profiles” on page 8.

Distributing Resources

You can distribute created custom resources (such as a sample, profile, DSL customization, custom diagram, or other) using Resource/Plugin Manager. For more information, see “Distributing Resources” in MagicDraw OpenAPI UserGuide.pdf.

Working with UML Profiles

Profile in UML is a special kind of Package, used to contain collections of stereotypes, domain specific data types, and libraries.

Creating Profile

Profile can be created in any Project, however often the same profiles are used among many projects. In order to reuse the same profile, it must be created as an independent file with shared data called *module*.

To create Profile as a module

1. Create a new project.
2. In the Browser, right-click the root “Data” model and create a new Profile.
3. Create a new Class diagram inside the Profile.
4. Create stereotypes in the diagram.
5. Use Extensions, Associations and Generalizations between them if needed.
6. In the Browser, from the created Profile shortcut menu, choose **Modules** > **Share Packages**.
7. Select Profile as shared, enter description, and click **OK**.
8. Save the project.
   Now you may use this profile in other projects.

To use the created profile in other projects

- From the **File** menu, choose **Use Module** and select the module file. In such a way only shared packages of the module will be loaded into the project.

TIP!	If the Profile contains diagrams, they will be also loaded in the user projects. If you don’t want to see these diagrams in your project, move diagrams from shared packages into root model or other non-shared packages in your module. In such a case diagrams with stereotype hierarchies will be available to users who are working directly on the profile creation, but not for users who are using this profile as a module.

Modifying Profile

If you want to make any changes in the profile, you may modify it by opening profile as a regular project or modify it in the read-write mode (see below).

Opening Profile as a project

This is the most natural and safe way to modify your profiles.

It helps to see all not shared infrastructure including diagrams, tests, examples and etc.

To open Profile as a project

- In the Browser, from the used Profile shortcut menu, choose **Modules** > **Open Module as Project**.
- On the **File** menu, click **Open** to open the Profile as a simple Project.

Modifying profile in the read-write mode

Using profile in the read-write mode is the fastest way to add changes when you are not in teamwork and do not care of the possible editing conflicts.

It helps to perform the basic refactoring of your modules by moving the elements directly from one module to the other.

You may choose read-write usage mode when trying to use some profile as a module.

To use Profile in the read-write mode

1. Do one of the following:
   - If the profile is not in use yet, on the **File** menu, click **Use Module**. Select a Profile and click **Next >**.
   - If the profile is already in use, from its shortcut menu, choose **Modules** > **Module Options**.
2. In the **Module Accessibility** area, click **Read-Write**.
   By setting this option you will be able to change the profile directly in your project.

Using profile, stereotypes and tags

The description of how to use profile in your project you may find in Section “Using Project Module”.

Applying stereotypes

A stereotype defines how an existing metaclass may be extended, and enables the use of platform or domain specific terminology or notation in place of, or in addition to, the ones used for the extended metaclass.

When a stereotype is applied to a model element, the values of the properties may be referred to as tagged values.

To apply stereotype to an element

- Select the element (in the Browser or on a diagram) and from its shortcut menu, choose **Stereotype**.
- In the element specification dialog box. Choose the stereotype from the Applied Stereotypes field or go to Tags and enter value for some tag. In this case stereotype will be applied automatically.
- Type stereotype name directly on element’s symbol.
  On the diagram pane in the element name area, type two angle brackets <<, type stereotype name and close angle brackets (or press Ctrl+Space and choose from auto-completion list). Then you may type element name itself.
  For example: if you want to name element “Books” and assign “<<table>>” stereotype, in the element name area, type the following: <<table>> Books.

To apply stereotype to multiple elements

- Select multiple elements and from their shortcut menu, choose **Stereotype** command or choose stereotype in the **Properties** panel below model tree.

Specifying tagged values

Special Stereotypes and Tags tree is used to reflect all stereotypes and tags that are applied or could be applied to this element.

To specify tagged values

1. Open element **Specification** dialog box.
2. Specify tagged values in the **Tags** pane.

In the **Tags** pane there are multiple filters that help to distinguish or group tags with different status.

To filter and group tags

-  Group by stereotype
-  Group by special tag groups (
-  Show only tags with values
-  Show type of tag definition
-  Show tags only from applied stereotypes

Using scalable stereotype icons

MagicDraw allows using scalable icons for stereotypes in such a way overriding standard UML notation.

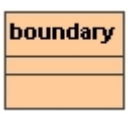
Good-looking scalable shapes are very important for scalable document formats (like *.pdf), presentations, and large printed diagrams.

To use custom large scalable icon instead of standard UML shape.

1. In the **Stereotype** specification dialog box, assign *.svg icon for stereotype.
2. Apply stereotype to some element in diagram.
3. Suppress content for this shape (suppress all possible compartments and select **Suppress content** presentation option in shortcut menu).
4. Resize the appeared icon to any size.

Example:

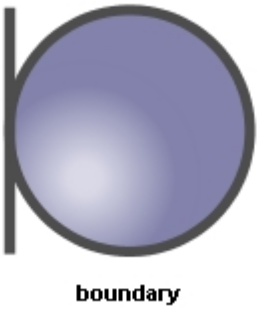
Before applying a stereotype:



After applying a stereotype (in such a case stereotype has its own style – blue color and icon):

After suppressing content:

Icon can be scaled up without the loss of quality:



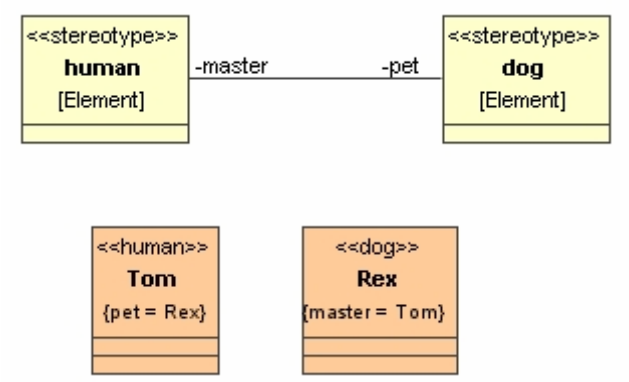
Advanced profiling

MagicDraw allows using several profiling enhancements that are not defined in UML, but that helps to solve some common problems in everyday modeling.

Initializing opposite association end

If two stereotypes are connected with Association and there are roles at both ends, role at one end will be initialized automatically when opposite role is specified.

Example:



When <<dog>> Rex is set as a pet for <<human>> Tom, Tom will be automatically set as “master” for Rex.

Invisible stereotypes, tags and constraints

Normally tags are used to hold some additional specific information that helps to interpret basic elements somehow different by some external tools, like transformations, code engineering, etc.

In these cases users don’t want to make impact on existing diagrams, only mark model.

Tag definitions are properties of stereotype in UML2, so UML doesn't allow using tagged values without assigning of stereotype. That means your diagrams will be changed when some tags are added into model.

MagicDraw provides ability to make chosen stereotypes or tags not visible in diagrams.

To make stereotype invisible

The stereotype, which you want to make invisible, must be inherited from <<InvisibleStereotype>> (from MagicDraw profile).

1. Create a custom stereotype.
2. Create Generalization between your stereotype and InvisibleStereotype

You custom stereotype will be not displayed on symbols where it will be applied. Note that all tags also will be invisible on symbols.

Example:

It is not desirable to see an author tag in diagrams. To accomplish this, follow these steps:

1. Create stereotype <<copyrighted>> that extends Element metaclass.
2. Create property "author: String" for <<copyrighted>> stereotype.
3. Inherit this stereotype from <<InvisibleStereotype>> (create generalization in the model or diagram).

Now you may enter tagged value "author" for any element in your model.

Stereotype <<copyrighted>> will be applied automatically but will be invisible in diagrams.

To create an invisible tag

Apply "InvisibleStereotype" on your tag definition (property of stereotype)

All such hidden tags will be invisible on symbols of stereotyped elements.

Example:

Stereotype <<identification>> has two tags: ID and Description. The user would like to see Description value on elements in diagrams, but ID should be hidden as it is used in some external model transformation tools or similar activities.

Apply <<InvisibleStereotype>> on ID property of <<identification>> stereotype and it will be not visible on stereotyped elements on diagrams.

To make a constraint invisible

A new capability to define invisible constraints has been introduced. To make a constraint invisible, apply the <<InvisibleStereotype>> stereotype to the constraint. Invisible constraints may be used while creating validation constraints, in DSL customization, or in other modeling cases.

Applying different Icons for the same stereotype

MagicDraw provides ability to create stereotype, which uses different icons, depending on stereotype "kind".

Example:

PseudoState is one in UML, but it uses different notation, depending on PseudoStateKind.

The similar situation could appear in your model.

To apply different icons on the same stereotype

1. Create Enumeration (in our example supercarKind). Enumeration will define all stereotype “kinds”.
2. Apply <<iconHolder>> stereotype on enumeration.
3. Create as many EnumerationLiterals as different kinds you would like to have (in our example it is ”ferrari” and ”audi”).
4. Apply stereotypes with different icons to every EnumerationLiteral (in our example <<ferrari>> and <<audi>>).
5. Create Stereotype (in our example supercar) with property (tag definition), which type is such enumeration (supercarKind). Default value could be selected.
6. Assign the created stereotype to an element and select one of enumeration value in tags. Stereotyped element will change its icon regarding enumeration value.

Example:

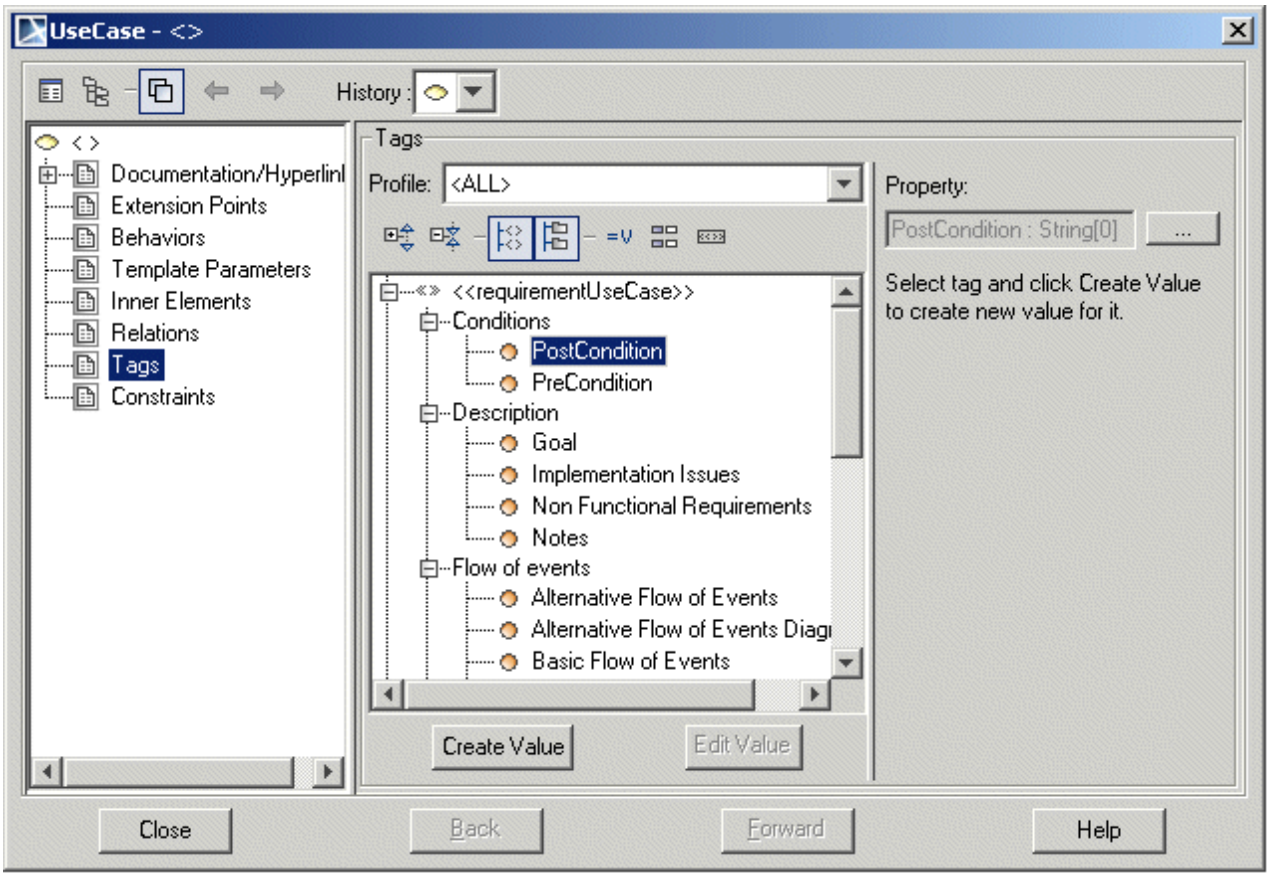
Grouping tags inside stereotype

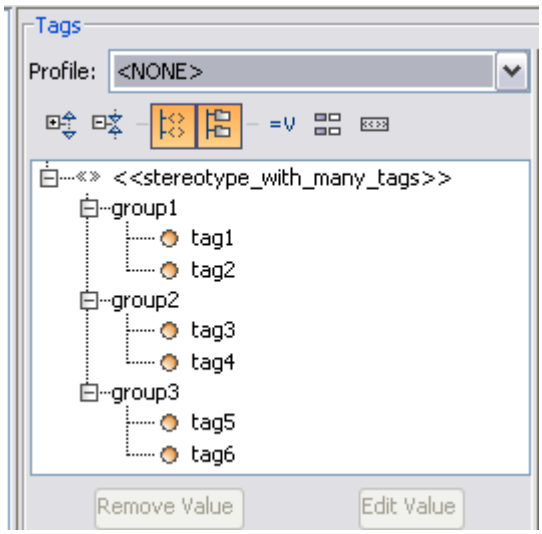
When some stereotype has many properties, it is very useful to group logically these properties into named groups. For this purpose every property could have tagged value with its group name.

To group tags

1. Open specification dialog of stereotype property.
2. Go to “Tags” panel.
3. Type group name as “groupName” value, << hasGroupName >> stereotype will be assigned automatically.
4. Repeat these step for every stereotype property

Example 1. Real example of grouped properties from UseCase Description Profile





Creating tags with default values

1. Create tag definition (Property).
2. Specify default value for this tag.
3. Specify multiplicity larger than 0 (1, 1..*, 2, etc.).

Such multiplicity means that this tag is mandatory.

When stereotype will be applied, mandatory tags will be automatically created and default values will be assigned.

Customizing style of stereotyped elements

There are several ways to customize symbol style of stereotyped elements.

To extend UML symbol styles by creating new styles for stereotyped elements

1. In the **Project Options** dialog box, the **Symbols Properties Styles** tree, right-click the **Stereotypes** branch. The list of stereotypes appears.
2. Select the check box near the stereotype and click the **Apply** button. Stereotype will be included into the **Stereotypes** branch. Set the stereotype style properties in the right pane of the **Project Options** dialog box.

The best practice is to create styles for stereotypes in the Profile, loading it as a project.

When some project uses Profile, styles for stereotypes will also be loaded and used. These styles will be applied right after applying stereotype on some element and will be used in all diagrams.

Style for stereotype in custom diagram

Custom diagram wizard allows specifying styles for stereotyped elements.

Note, that this style will be used only in diagrams of this type.

For more details of how to work with Custom Diagram Wizard, see the next chapter.

Custom Diagram Wizard

Custom diagram wizard is a powerful engine that allows creating your own diagram types for specific domain, platform, technology, or other purposes.

It allows you to create your own custom elements in diagram toolbar, custom symbol styles and other customizations.

To open the Customize Diagrams Dialog box

- From the **Diagrams** menu, choose **Customize**.

You may change properties of existing diagrams (Edit function) or create your own brand new diagram type (Create function). Diagram customization descriptors are saved into separate file for every diagram, so you are able to exchange these customizations with your partners or colleagues (use Import or Export function).

Reset to default button in **Customize Diagrams** dialog box restores default configuration for diagrams bundled with MagicDraw installation (it does not work with user defined diagrams). Click **Edit** or **Create** to open **Custom Diagram Wizard**.

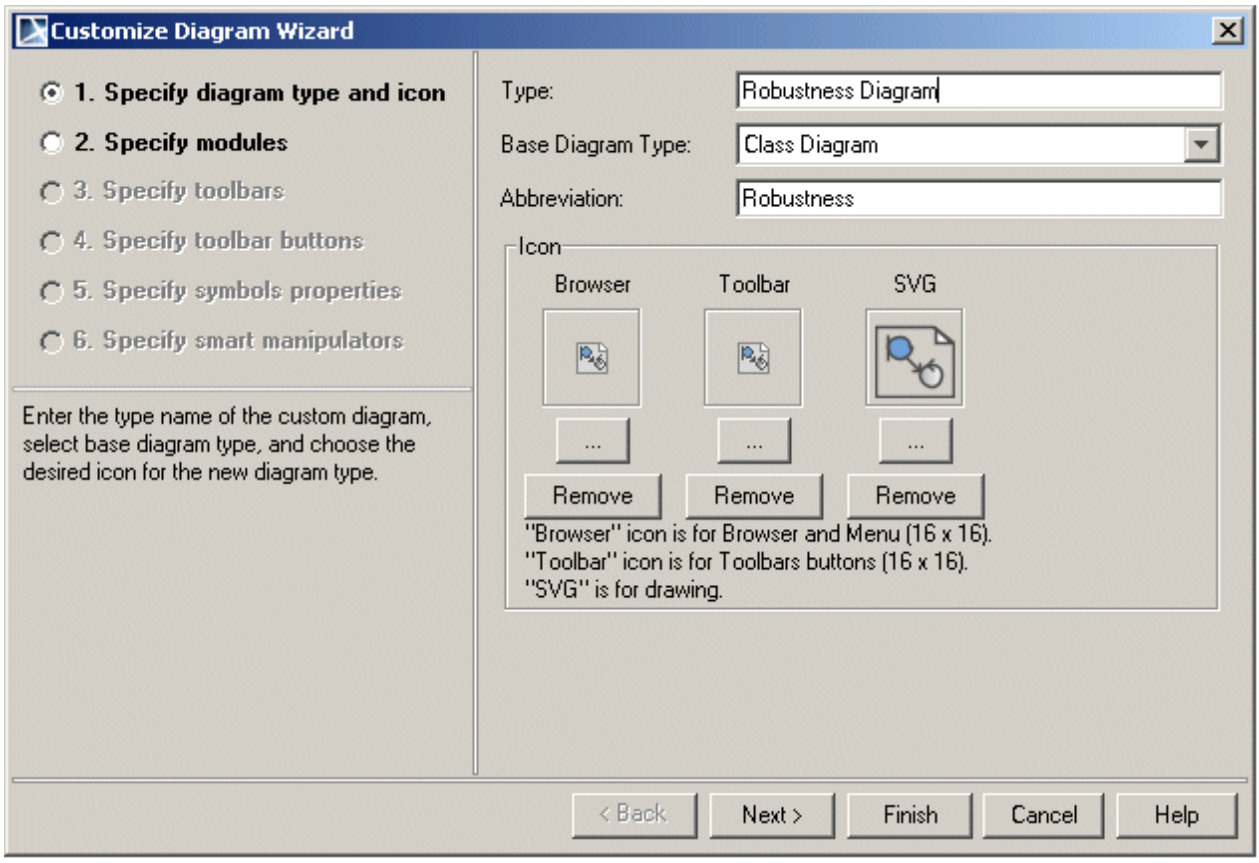
All custom diagram wizard steps with explanations are described in the chapters below.

Name, base type, abbreviation

All Custom Diagrams are based on some standard UML diagrams (like Class, Use Case, Sequence etc.).

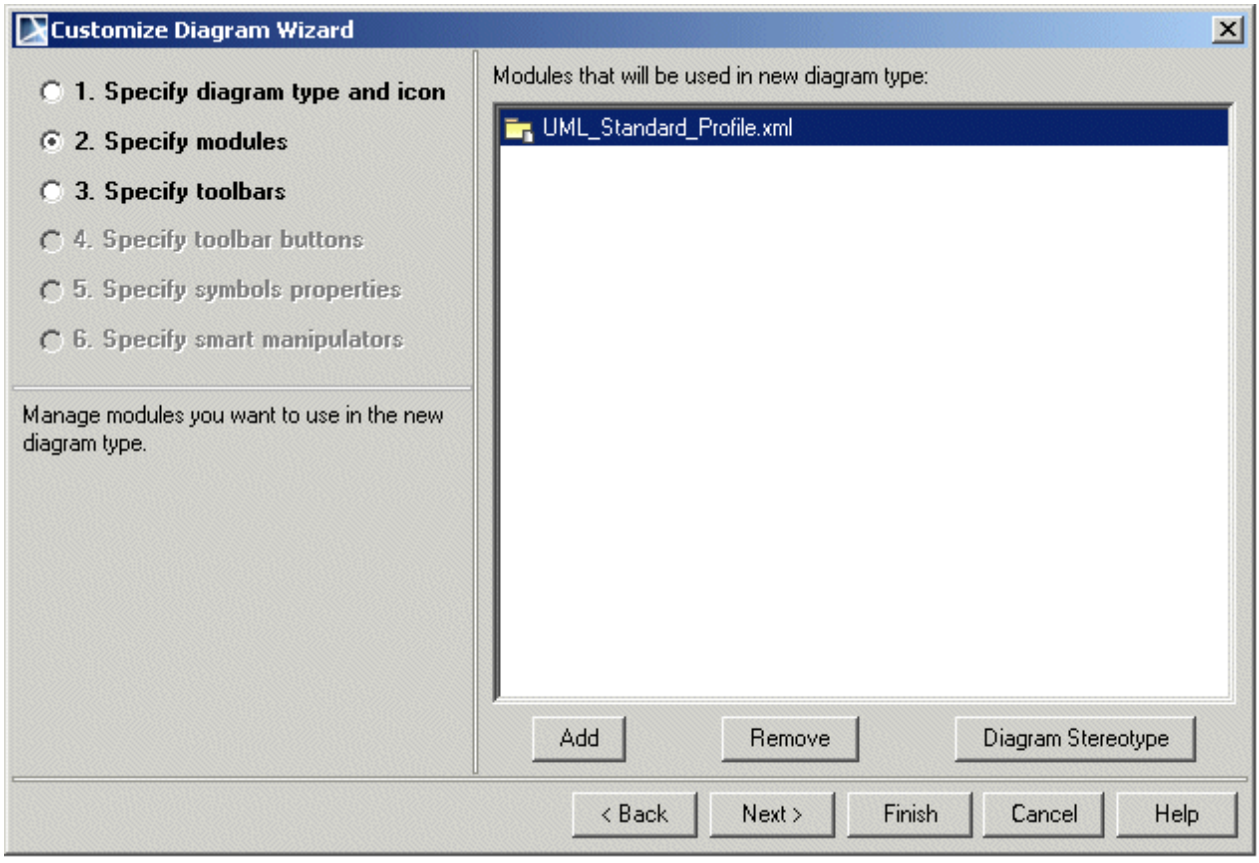
Start to create your own diagram from these steps:

- Diagram type name (e.g. Robustness Diagram)
- Base Diagram type – standard UML diagram which is extended.
- All configurations, semantics and other settings will be inherited from this diagram type.
- Abbreviation – short form of diagram name. It will be used in Diagram Frames header or Diagram shapes used in Content diagrams.
- Icons – several icons for your custom diagram representation in MagicDraw GUI.



Used profiles

Custom Diagrams are oriented to some new specific domain, technology or platform and are often based on some custom profile.



In the second Wizard step (Specify Modules) you may select required modules or profiles.

Custom diagram could use stereotyped elements, so profiles that define these stereotypes must be used by custom diagram.

The selected modules/profiles will be loaded when custom diagram will be created in user projects.

By default UML Standard Profile is used (and must be used).

You may choose stereotype for diagram itself by clicking the **Diagram Stereotype** button.

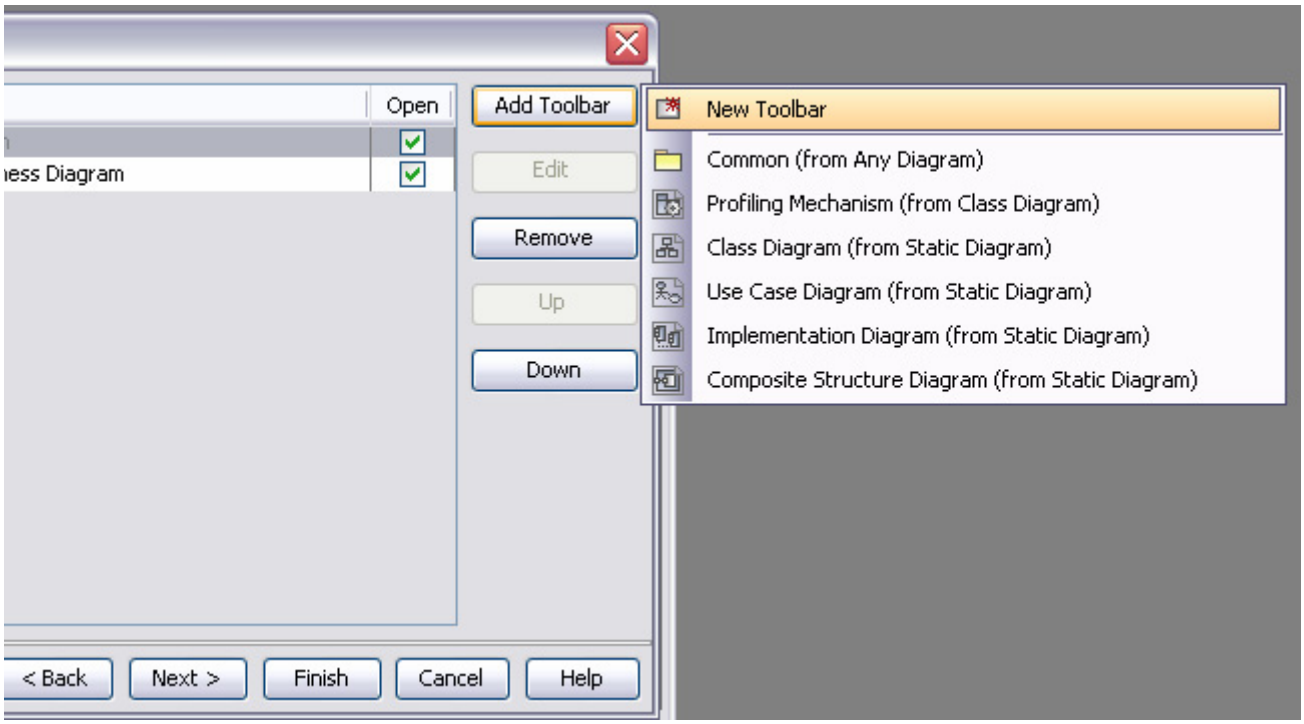
Custom diagram toolbars

Every diagram differs by elements that are used in these diagrams. In the **Specify toolbars** tab, you may group standard or custom elements in any toolbars or order you want.

You may:

- Create your own custom diagram toolbar
- Create your own toolbar, name it, and select the icon.
- Choose standard toolbars that will be visible in your diagram.
- Select existing toolbars inherited from base diagram type (e.g. see UseCases toolbar in Class diagram subtype).

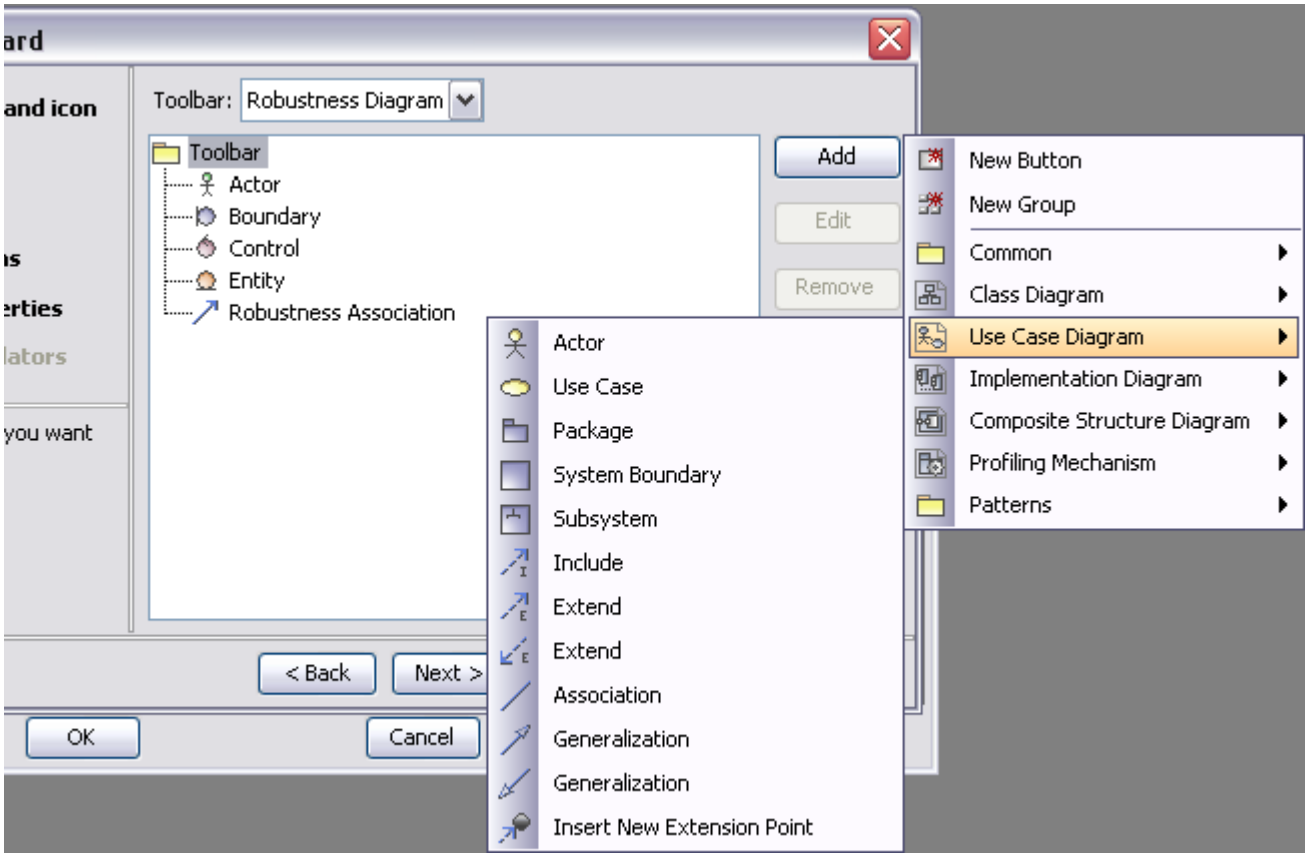
- Order toolbars using “Up” and “Down” buttons
- Select which toolbars will be expanded or collapsed by default (use “Open” checkbox)



In this step you may manipulate with toolbars, in the next step you will be able to customize buttons inside selected toolbars.

Custom toolbar buttons

In the **Specify toolbar buttons** tab, select standard or stereotyped elements for your own diagram toolbars.

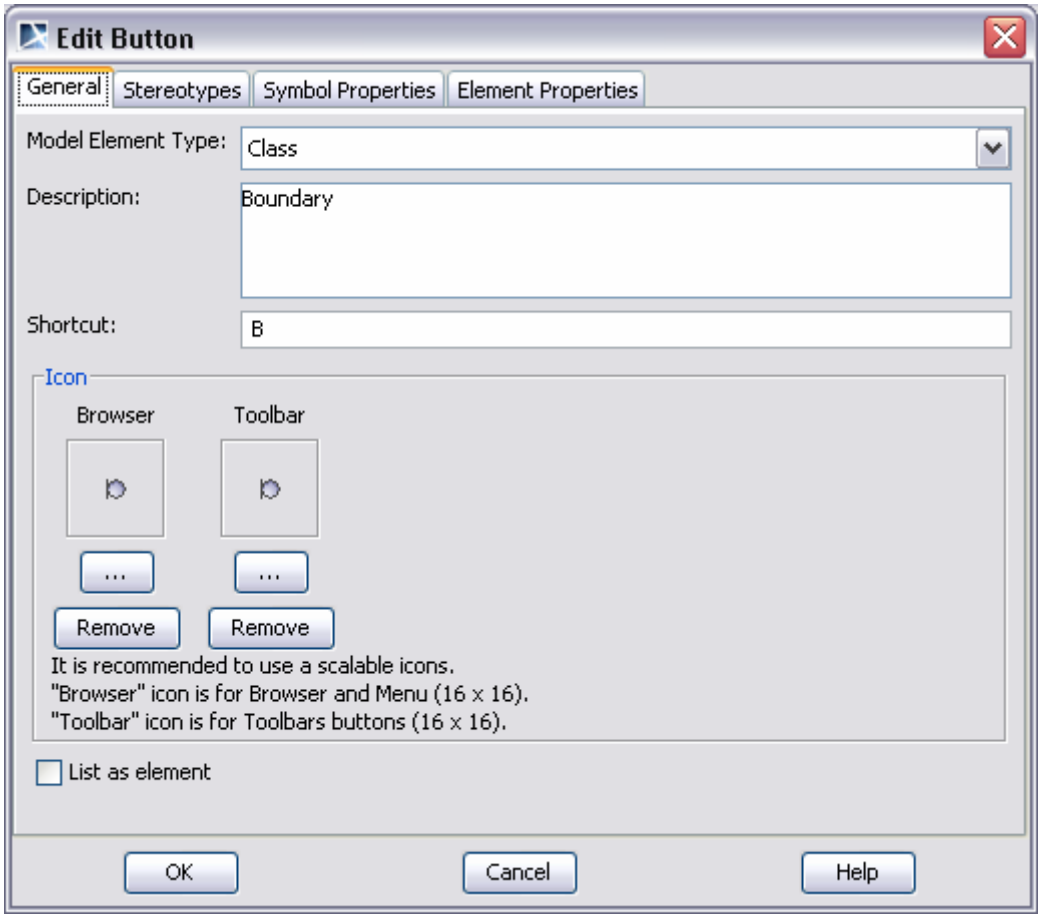


Click **Add** and select standard UML elements or click **Add** and then **New Button** to create your own buttons for your own customized or stereotyped elements creation.

Customize the following properties when creating the new button:

- Model element type.
- Custom icon for button.
- Shortcut that activates this button.
- Description that will be visible in the tooltip.
- Stereotype(s) that will be applied for created element.
- Symbol style. The selected symbol properties will be used only when element will be created using this button.

- Default values for some primitive model element properties (like isAbstract = true or similar)



Custom styles

In the **Specify symbols properties** step, you may customize style for any element that appears in your custom diagram (e.g. class dropped in your diagram should be suppressed and red).

You may change the appearance of standard symbols, symbols of stereotyped elements, and custom diagram itself.

Customized element style will be used only in the appropriated custom diagram type.

Smart manipulators

Smart manipulators are special small buttons that appear in the popup window when symbol is selected in diagram.

In the **Specify smart manipulators** step, you may configure what kind of Relationships will be suggested when custom element is selected in diagram.

There are three major steps:

1. Create new configuration (or modify existing one).Select element you want to be customized. Smart manipulators configuration can be related to:
    - Element (displayed as [Element name])
    - Element + stereotype(s)
    - Symbol (displayed as {Symbol name})

- Symbol + stereotype(s)
- Stereotype(s) (displayed as <<Stereotype name>>)

2. Select suggested relationships for the selected configuration.
3. Select target elements for the selected relationship.

If few configurations could be applied for the same selected element in diagram, only one configuration will be used by such priority**:**

#	Configuration	Comment
1	Symbol + stereotype(s)	When stereotype(s) for symbol applied
2	Stereotype	
3	Symbol	
4	Element + stereotype(s)	When stereotype(s) for element applied
5	Element	

NOTE If there are created configurations for few stereotypes (ex. Stereotype1 and stereotype2) in the same diagram, after both stereotypes will be applied for one Symbol on that diagram pane, first configuration (in this ex. Stereotype1) will be used, if there are no symbol+ stereotype(s) configuration specified.

Inheritance of configurations

All diagrams have base diagram from which they inherit configurations.

For example there is such hierarchy: Any diagram->Static diagram->Class diagram->Generic DDL diagram

If you add new configuration to Class diagram (ex. Symbol A + stereotype B) this configuration will be used in Generic DDL diagram as Class diagram is its base diagram. Any and Static diagrams will not have such configurations.

To change configuration from the base diagram type

- From the **Configurations** list, select configuration by element type and choose **Specify own configuration** option button. Inherited configuration will be overwritten.

DSL Customization engine

DSL customization is a model-driven approach, based on UML profiling. A special DSL Customization Profile is used for this purpose. This section explains how stereotypes from DSL Customization Profile should be used to customize your own domain specific profile.

General principles

The MagicDraw DSL customization engine is able to process user-defined rules for DSL elements and reflect this in MagicDraw GUI and diagrams behavior.

All customization information is stored as special tagged model.

There are two ways for creating and using this customization data:

- Create customization model in a DSL profile, in a separate package.
  This way is the most intuitive and comfortable, but the profile will be polluted (additional package will be added). It is not acceptable when profile shall be standard compliant
- Create customization as separate module file. This customization module shall use original DSL Profile.
  This way helps to avoid profile pollution, but is more confusable, because “customization module” shall be used in projects instead of the “original profile”. The “original profile” will be used indirectly.

In this case all user projects shall use customization module instead of custom DSL profile. Custom DSL profile will be used as secondary module automatically.

Creating customization data

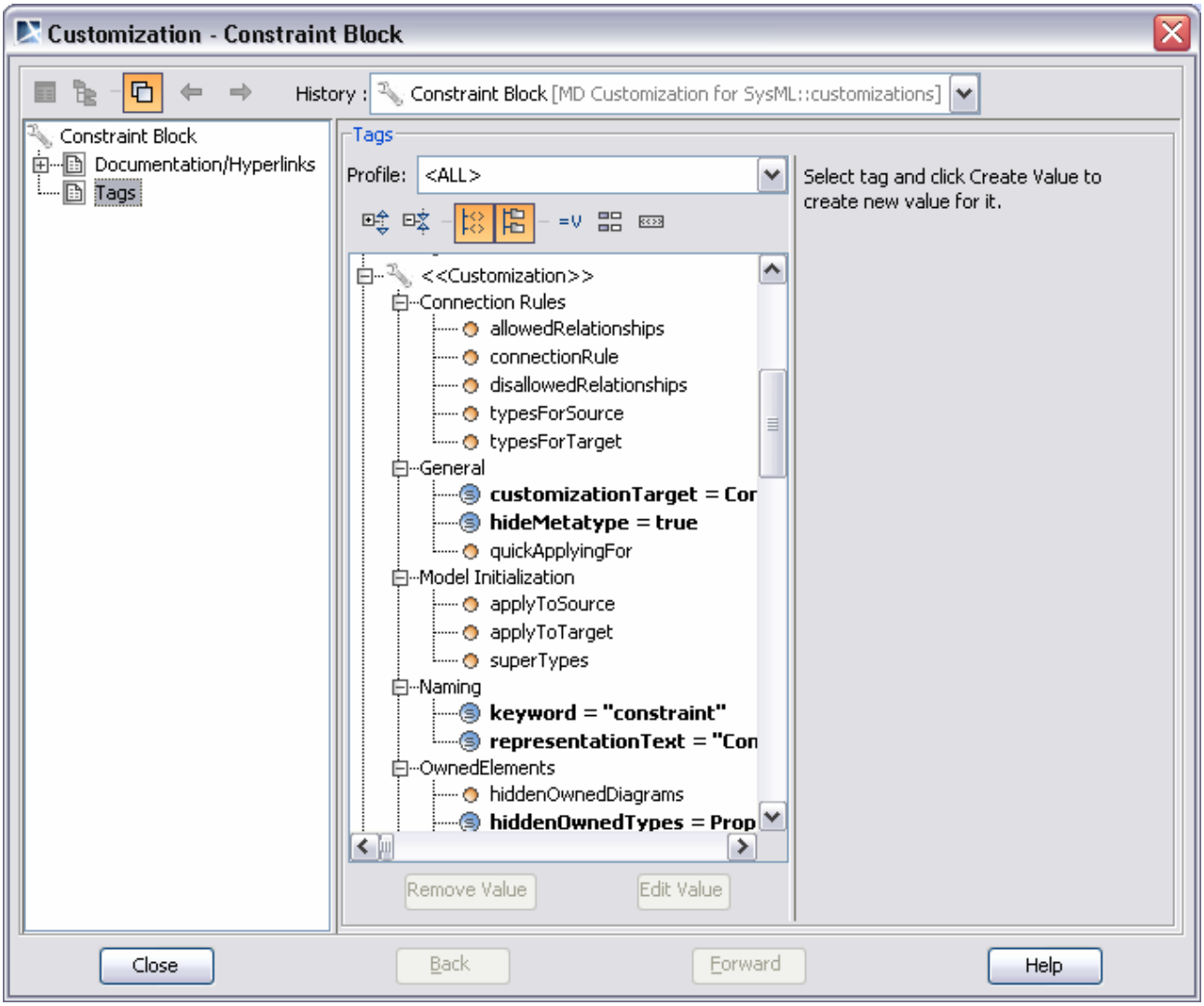
All DSL customization rules are stored as tag values in Classes, marked with <<Customization>> stereotype.

These classes will be parsed and interpreted after reloading the project.

To create customization data

1. Create a package for customization data.
2. Create a class in the package (or a class in a class diagram of this package).
3. Assign <<Customization>> stereotype to this class.

4. Specify tag values for the <<Customization>> stereotype.



In general, each customization class has the following mandatory tags:

1. **customizationTarget** – a stereotype, which you are going to customize.
2. **hideMetatype** – if true, the stereotype acts like a new standard element type in MagicDraw.

Example:
If <<Block>> stereotype for class is customized with “hideMetatype=true”, all stereotyped classes will be interpreted as Blocks, but not as Classes. The UML Class underneath will be hidden.

The other tags are described in the subsequent sections that are organized by customization purpose.

Defining preferred metatype

DSL customization engine allows for specifying a preferred metatype, if there is more than one metatype defined for the DSL type (the term is defined in the section “Introduction” on page 5 of this user guide).

To define a preferred metatype for the DSL type

1. Select the appropriate customization class and open its Specification window.
2. Click the **Tags** tab.

3. In the **Tags** specification pane, select the preferredMetatype tag and click the **Create Value** button.
4. In the element Selection dialog, select a preferred metatype and click **OK**. The selected metatype will be assigned as the preferredMetatype tag value (see the figure below).

IMPORTANT!	Multiple selection is not allowed. Define a single value.

5. Click **Close**.

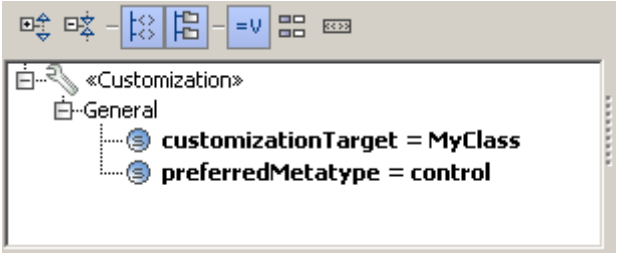


Figure 1 -- The preferredMetatype tag with the value defined

NOTE	After updating the customization class specification, either the DSL Customization profile must be reused in the model or the model must be reloaded.

From now on all the elements of this DSL type will be created with the preferred metatype, which has been specified in the DSL type customization.

For the detailed information on how to create a customization class, please refer to the section “Creating customization data” on page 22 in this user guide.

Creating your own property groups and subgroups

You can create your own groups and subgroups to display the properties either of standard UML metaclasses or stereotypes. You can also choose, where you want these property groups and subgroups to be visible. Property groups and subgroups can be visible in the following places:

- Element’s Specification window.
- Element’s **Properties** panel (at the bottom of the Model Browser).
- Element’s shortcut menu > **Go To** > submenu.
- **Compartment Edit** dialog.

- **Criterion Editor** dialog for editing the relation's criteria in the Relation Map diagram.

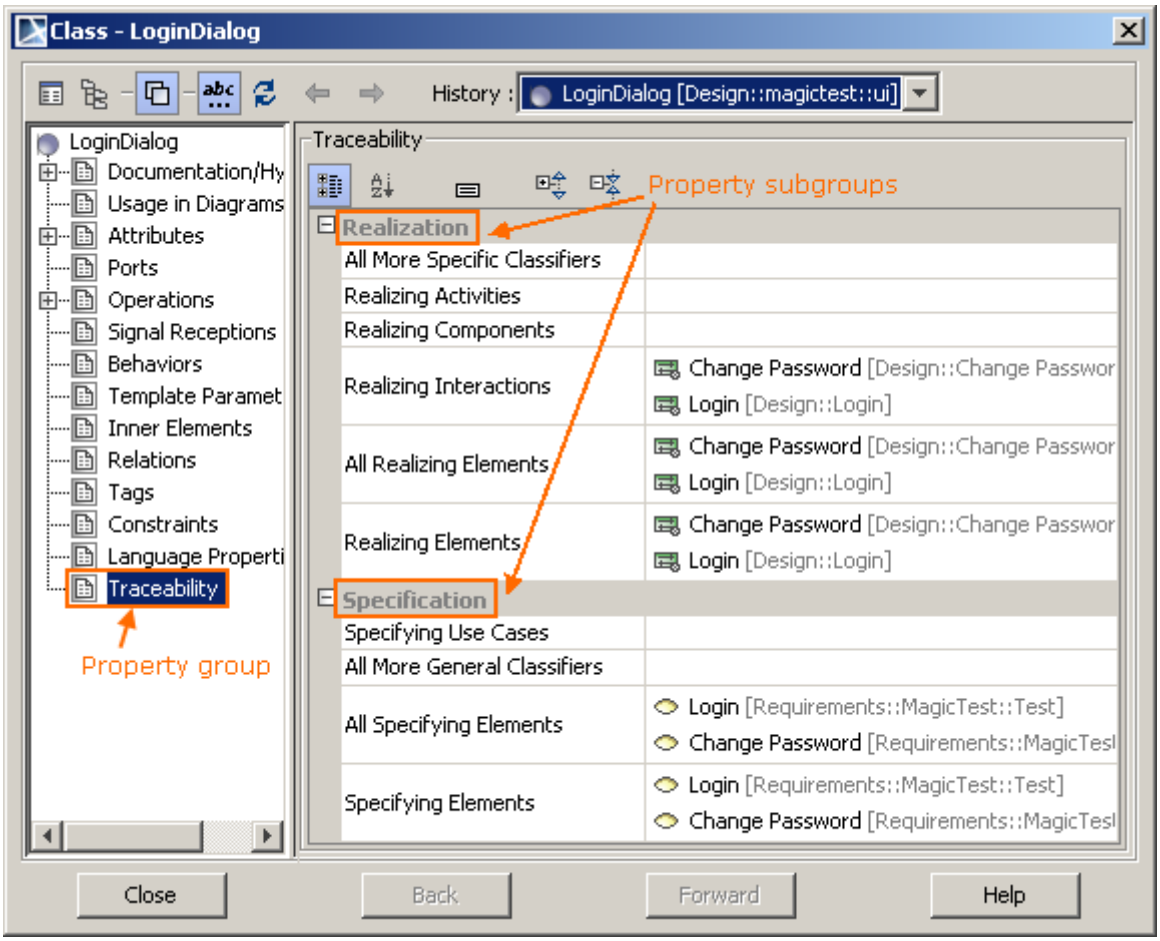


Figure 2 -- Property groups and subgroups in the element's Specification window

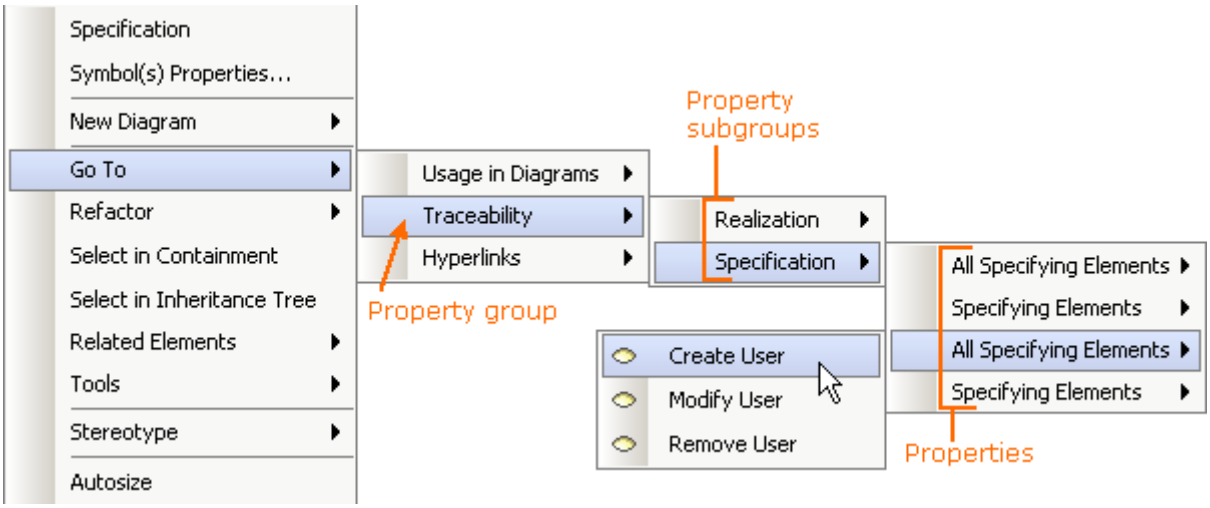


Figure 3 -- Property groups and subgroups in the element's shortcut menu

To create a property group

1. Right-click a customization class and choose **New Element** > **Property Group**.
2. Type the attribute's name (it will be used to name the property group).
3. Open the attribute's specification and click the **Tags** tab.
4. In the **Tags** specification pane, edit the tag values (tags are described in the table below).

5. Click **Close**.

A stereotype <<propertyGroup>> has been automatically applied to the newly created attribute.

To create a property subgroup

1. Choose the customization class, for which you want to create a property subgroup.

IMPORTANT!	The class must have at least one attribute with the stereotype <<propertyGroup>> applied.

2. Right-click the attribute with a stereotype <<propertyGroup>> and choose **New Element** > **Property Group**.
3. Type the attribute's name (it will be used to name the property subgroup).
4. Open the attribute's specification and click the **Tags** tab.
5. In the **Tags** specification pane, edit the tag values (tags are described in the table below).

IMPORTANT!	A subgroup does not inherit the tag values of the group. You have to specify them for each subgroup separately.

6. Click **Close**.

A stereotype <<propertyGroup>> has been automatically applied to the newly created attribute.

NOTE	The stereotype <<propertyGroup>> is applied in both cases: either creating a property group or a subgroup.

Whether the attribute is a property group or a subgroup, depends on the attribute hierarchy:

1. If an attribute with a stereotype <<propertyGroup>> is owned by a customization class, then it is a property group.
2. If an attribute with a stereotype <<propertyGroup>> is owned by another attribute with the same stereotype, then it is a property subgroup.

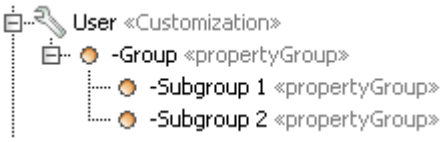


Figure 4 -- An example of customization class with the attributes for property grouping and subgrouping

The following table gives the descriptions of the stereotype <<propertyGroup>> tags.

Tag	Description
showGroupInCompartmentEdit	This property allows for displaying the element properties in the **Compartment Edit** dialog > the **Element Properties** tab and in a note on a diagram either within the property group/ subgroup or without it. If the value is true, the property name will be displayed as follows: *<property_group_name>::<property_name>*. If the value is false, the property name will be displayed as follows: *<property_name>*.
showGroupInElementSpecification	If the value is true, the property group/ subgroup will be visible in the element's Specification window.
showGroupInGoTo	If the value is true, the property group/ subgroup will be visible in element's shortcut menu as a submenu under the **Go To** menu.

Tag	Description
showGroupInQuickProperties	If the value is true, the property group/ subgroup will be visible in the element's **Properties** panel.
showGroupInRelationMap	If the value is true, the property group/ subgroup will be visible in the **Criterion Editor** dialog, when you create or edit a Relation Map diagram.
useAsNode	If the value is true, the property group will be displayed in a separate tab in the element's Specification window and/ or **Properties** panel. If the value is false, the property group will be displayed as a group in the general (default) pane of the element's Specification window and/ or **Properties** panel. **NOTE:** Do not set this tag value to true for the property subgroup.
properties	Stores a list of properties that will be visible in the property group or subgroup. **NOTE:** If the group has at least one subgroup, it may have no properties.
columns	Stores the specified table columns for showing the values of a multivalued property in the element's Specification window, for example, ownedOperation in Class. NOTES: • Do not specify this tag, when there is more than one property assigned to the properties group and/ or subgroup, i.e., when there is more than one value added to the "properties" tag. • The property group, to which the multivalued property is assigned, must be specified as a separate tab in the element's Specification window, i.e., the "useAsNode" tag must be set to true.
filter	Stores the element types (UML metaclasses and custom stereotypes) that will be displayed as property values.

Example:

Let's create a group "My Group" in the element's **Properties** panel. This group will be displayed as a new tab and its properties will be grouped into the subgroups "Subgroup1" and "Subgroup2".

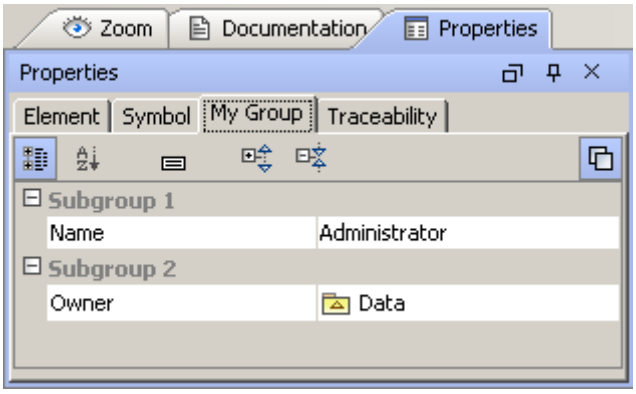


Figure 5 -- The property group "My Group" in the element's Properties panel

The element's customization class will have the property representing the property group.

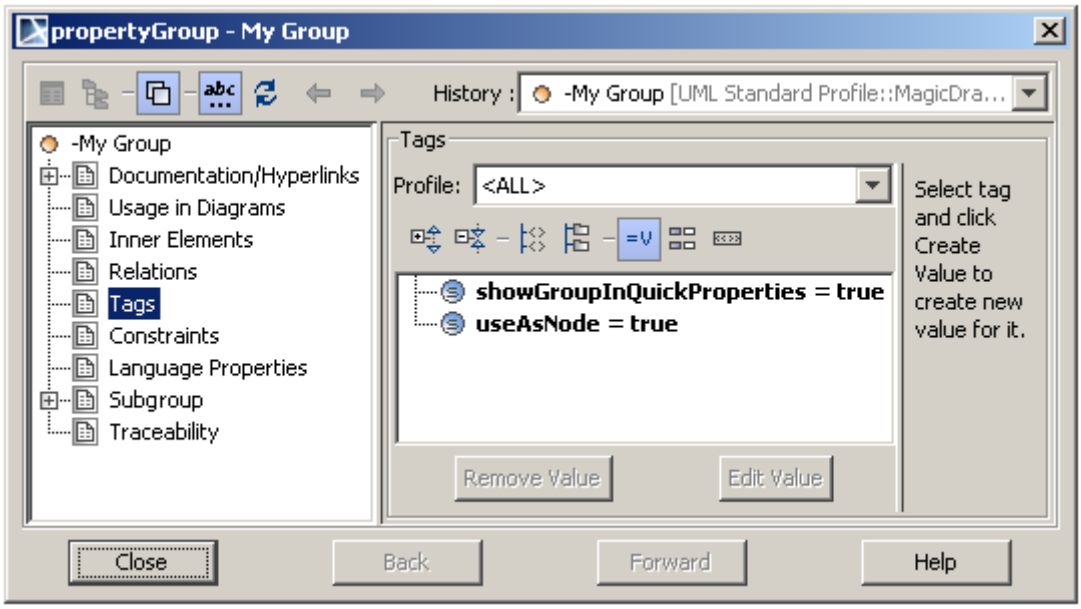


Figure 6 -- The Tags tab of the property "My Group"

The property group will have another two properties representing the property subgroups.

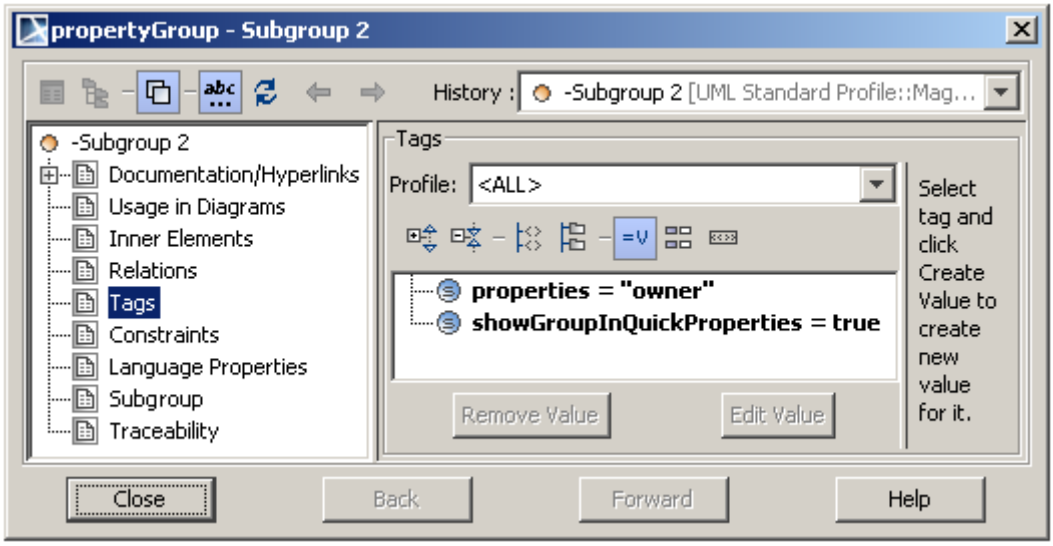


Figure 7 -- The Tags tabs of the properties "Subgroup 2"

Merging property groups and subgroups

You can specify property groups and subgroups for different customization classes of the same element. The element, to which these customizations are applied, will have groups and subgroups from all the customization classes.

Example:

Let's say we have two customizations of the same element type, the actor. The first customization has the property group "My Group" with the subgroup "Subgroup 1" containing the property "Prop 1". The second customization has the same property group with the same subgroup specified, but the subgroup "Subgroup 1" contains the property "Prop 2".

As a result, the actor element type will have properties group “My Group” with the subgroup “Subgroup 1” containing both properties “Prop 1” and “Prop 2”.

Default visibility of property groups and subgroups

You can create property groups and subgroups for both standard UML metaclasses and stereotypes. Whether the group will be added to the metaclass or to the stereotype, depends on the customization target of the customization where the property group is created.

If the customization target is a stereotype and the “hideMetatype” tag is set to true, then the property group specified in the customization will be visible only for the elements with this particular stereotype applied.

If the customization target is a metaclass, then the group specified in this customization will be visible for the elements of this particular type. This group will also be visible for the elements with the stereotypes applied, except for the cases when the “hideMetatype“ tag in this stereotype customization is set to true.

The table below lists the default property group visibility in the element’s specification depending on the DSL customization.

Customization target	**Property group visibility in element’s Specification window and Properties panel**		
	Element of metaclass type	**Element with stereotype which DSL customization “hideMetatype=false”**	**Element with stereotype which DSL customization “hideMetatype=true”**
Stereotype	Invisible	Visible in standard mode	Visible in standard mode
Metaclass	Visible in standard mode	Visible in standard mode	Visible in all modes

Customizing Specification window

The main purpose of DSL customization is to hide UML, so extensions would look in MagicDraw as standard elements, but not stereotyped UML elements.

Domain specific element properties will appear in the Specification dialog and the **Properties** panel as regular properties, but not as tags.

There are several customizations and grouping cases for properties.

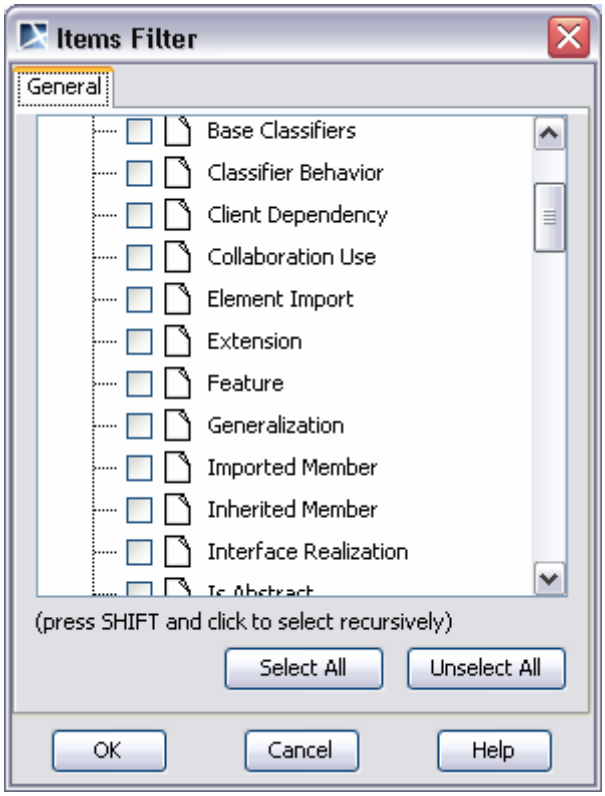
Hiding standard UML properties

DSL element specification may contain extended UML element properties and custom properties (tags) defined in stereotype.

All extended element properties will be used by default, but you may select which extended UML element properties should be used and which hidden in customized specification dialog.

1. **usedUMLProperties** tag of <<Customization>> stereotype is used for this purpose.
2. Open customization class, go to the Tags specification pane and create a new value for the “usedUMLProperties” tag.

3. Special editor will let you to select UML properties of extended UML element.



Properties can be hidden using Standard/Expert property modes also.

Customizing Standard/Expert property mode

You may save the default configuration for every stereotype inside Customization model, “standardExpertConfiguration” tagged value, using the same editor as in Specification dialogs, “Customize” functionality.

Dialog contains list of used UML properties + custom properties (tags), so user is able to customize which UML properties and DSL specific properties should appear in specification dialogs.

Property groups (nodes) could be customized also, they appear at the end of list, undesirable nodes could be hidden.

Always visible properties

Generally stereotype properties become visible in an element’s Specification window only after the stereotype is applied on the element. However, you may customize stereotype properties to be visible in the element Specification window even if the stereotype is not yet applied on the element.

This feature allows specifying some domain-specific properties for standard UML elements. It has already been used in MagicDraw to make stereotype properties, such as **Type Modifier**, **Active Hyperlink**, and **To Do** visible even if appropriate stereotypes are not yet applied on an element.

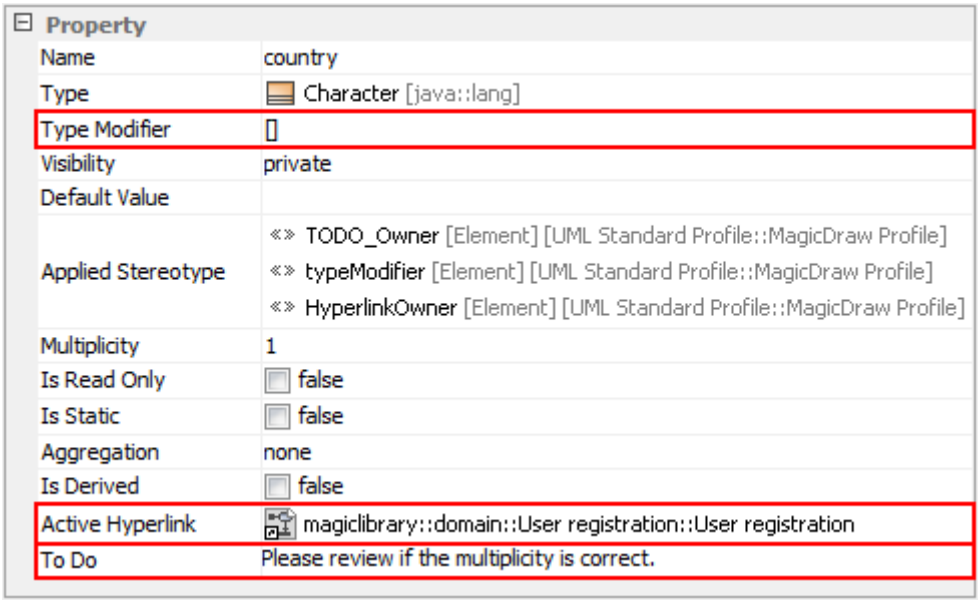


Figure 8 -- Always visible properties in UML property Specification window

To make stereotype properties visible, when the stereotype is not yet applied on an element

1. Open the **Tags** tab in the Specification window of the stereotype's customization class.
2. Set the **showPropertiesWhenNotApplied** tag value to *true*.
   All properties of the customized stereotype will become visible in the element Specification window even if the stereotype is not yet applied on the element.

Example:

Use the C# Profile (*C#_Profile.mdzip*) in your project and you will see an additional tab, the **C# Language Properties** one, appeared in a class Specification window. C# properties appear in the Specification window of a class even though no profile-specific stereotype is yet applied on the class.

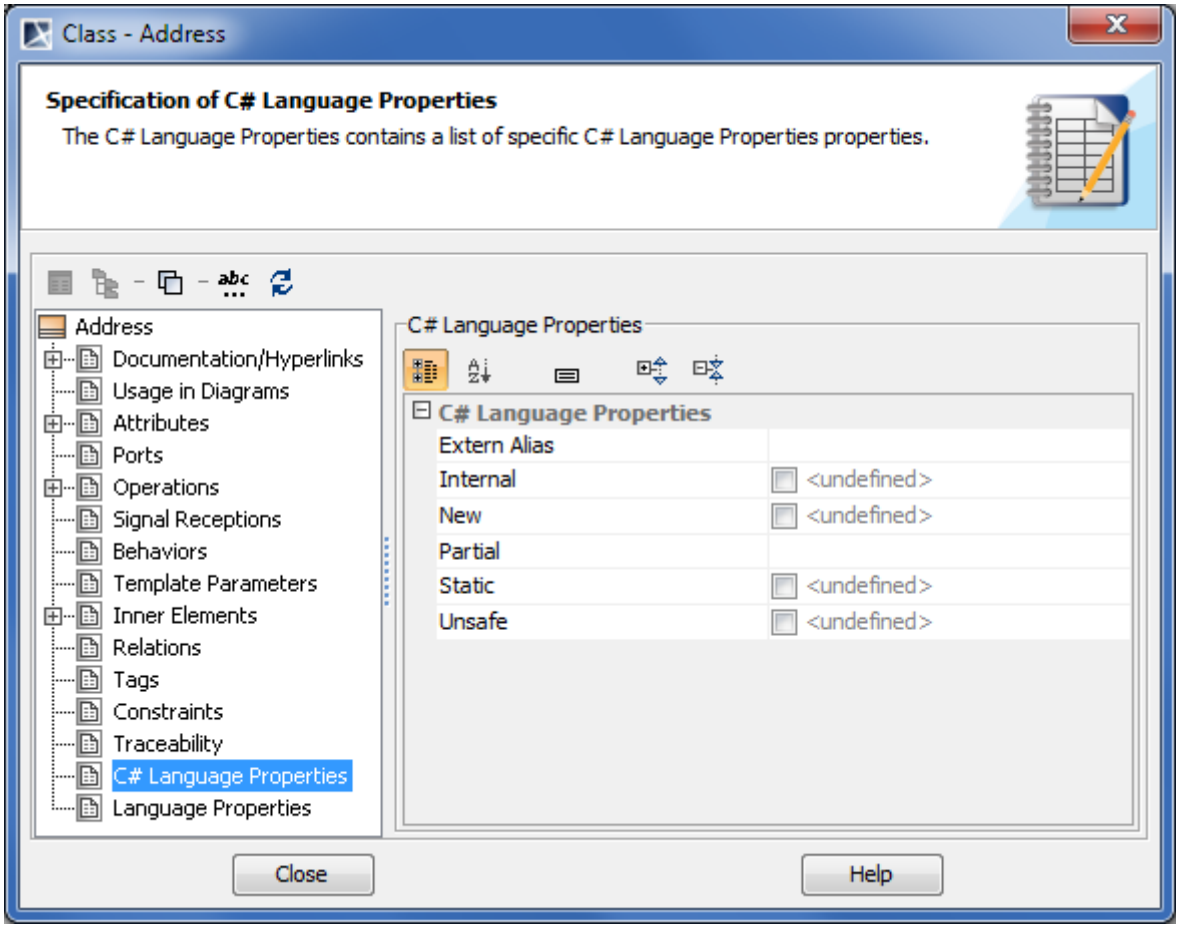


Figure 9 -- C# properties in class Specification window

Specify a desired C# property value in the class Specification window and you will see the «C#LanguageProperty» stereotype automatically applied on the class.

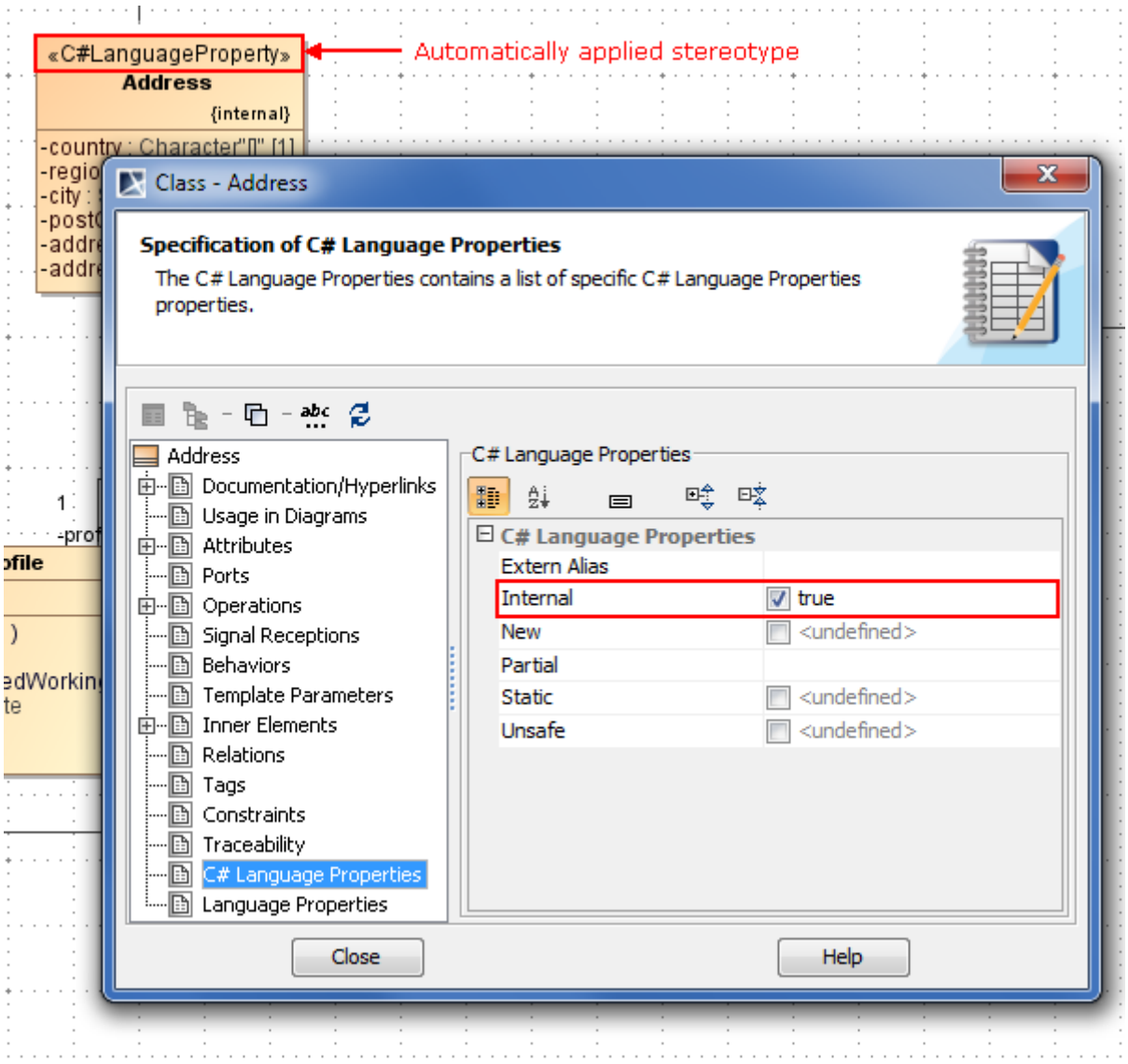


Figure 10 -- Class with automatically applied «C#LanguageProperty» stereotype

NEW! Advanced cases of properties visibility

There are two cases of displaying not yet applied stereotype properties. You can customize the stereotype properties visibility to depend on:

- **Profile application.** Properties will be visible only when the owning package of an element has a corresponding profile application.
- **Another stereotype and / or metaclass.** Properties will be visible only when:
  - Some other stereotype is already applied on an element.
  - Element's metaclass is one of the metaclasses extended by the stereotype.

To make the visibility of stereotype properties dependent on a corresponding profile application

1. Open the **Tags** tab in the Specification window of the stereotype's customization class.
2. Set the **showPropertiesWhenNotApplied** tag value to *true*.
3. Set the **showPropertiesWhenNotAppliedLimitedByProfileApplication** tag to *true*.

To make the visibility stereotype properties dependent on another stereotype or a metaclass

1. Open the **Tags** tab in the Specification window of the stereotype's customization class.
2. Set the **showPropertiesWhenNotApplied** tag value to *true*.
3. Select a stereotype or a metaclass as the **showPropertiesWhenNotAppliedLimitedByElementType** tag value.

TIP!	You can select more than one value.

Example:

Let us analyze the customization class of the «OraUser» stereotype in the Oracle Customization profile (*Oracle_Profile.mdzip*). This profile belongs to the Cameo Data Modeler plugin and can be used for database modeling.

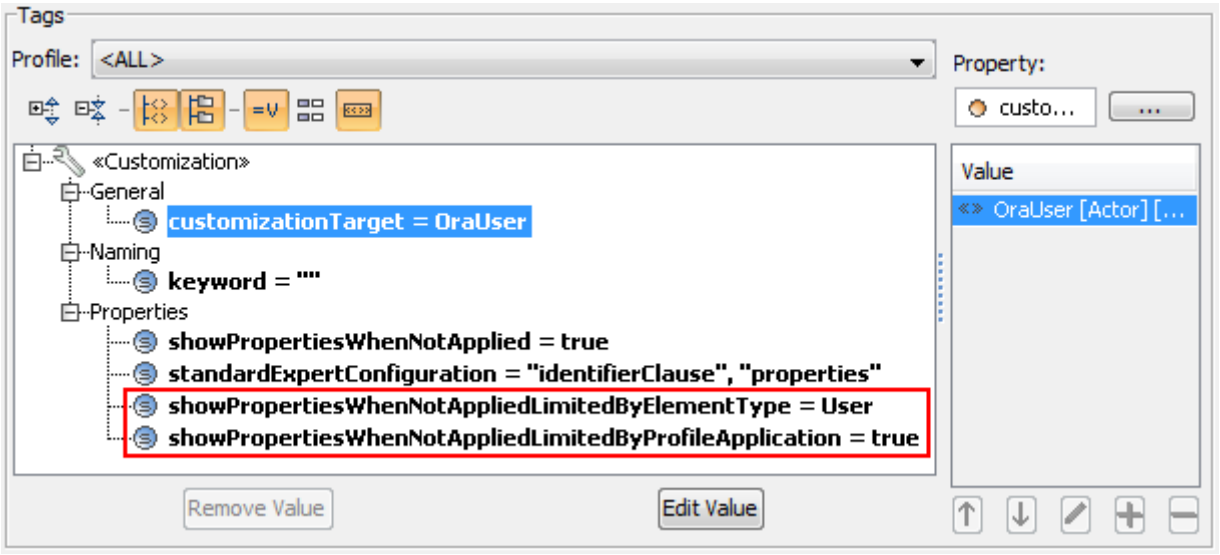


Figure 11 -- Tags for specifying advanced cases of properties visibility in «OraUser» stereotype customization class

As you can see in the preceding figure, values are set for both the **showProperties...LimitedByElementType** and **showProperties...LimitedByProfileApplication** tags.

The «User» stereotype is selected as the **showProperties...LimitedByElementType** tag value. This means that properties of the «OraUser» stereotype will be visible only for elements those have the «User» stereotype applied.

The **showProperties...LimitedByProfileApplication** tag value is set to *true*. This means that properties of the «OraUser» stereotype will be visible only for elements, whose owning package has the Oracle Customization profile application, i.e., for elements inside the Oracle schema.

In conclusion Oracle-specific properties will be visible only for users inside the Oracle schema.

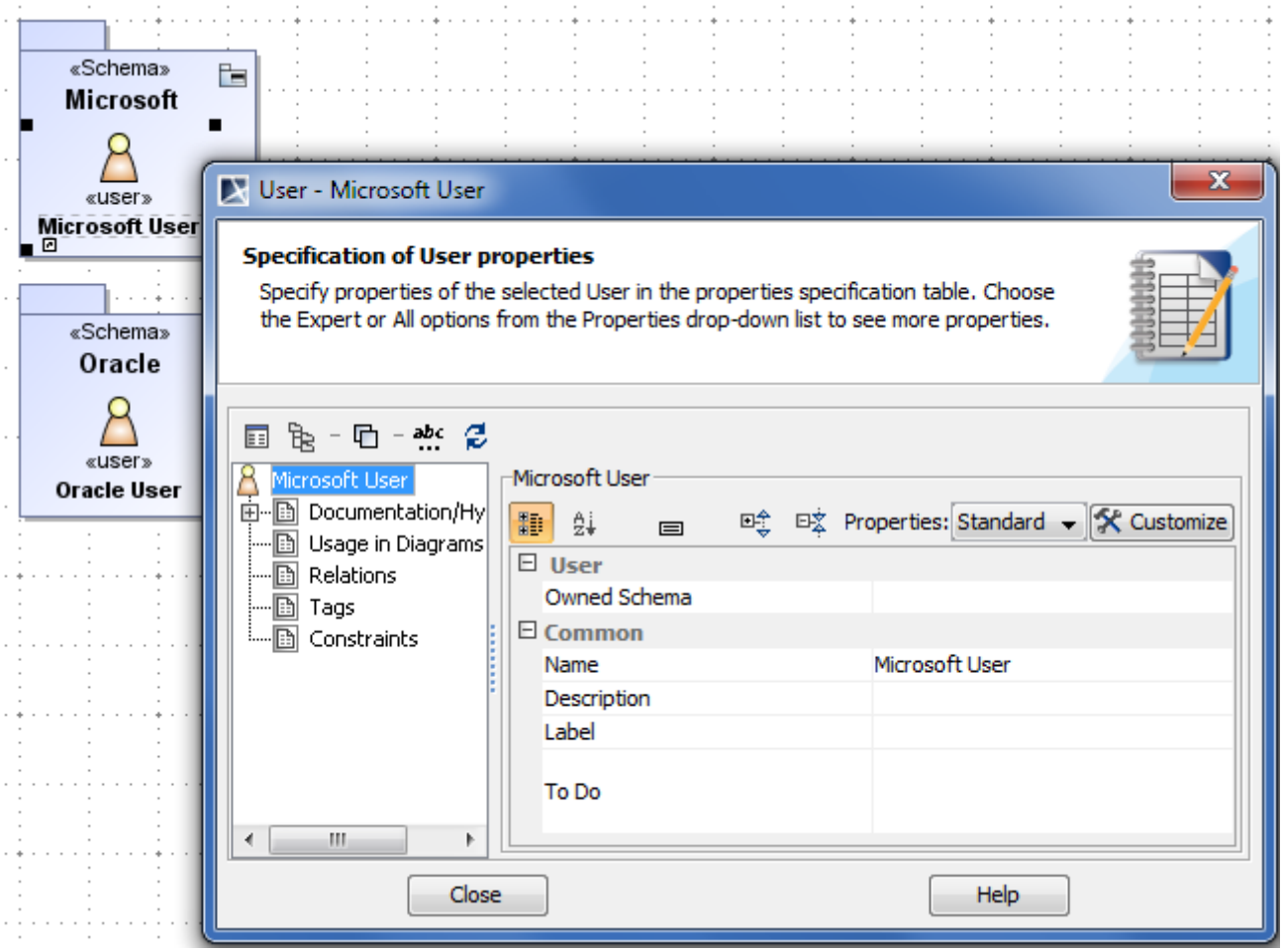


Figure 12 -- Specification window of user that is NOT INSIDE Oracle schema

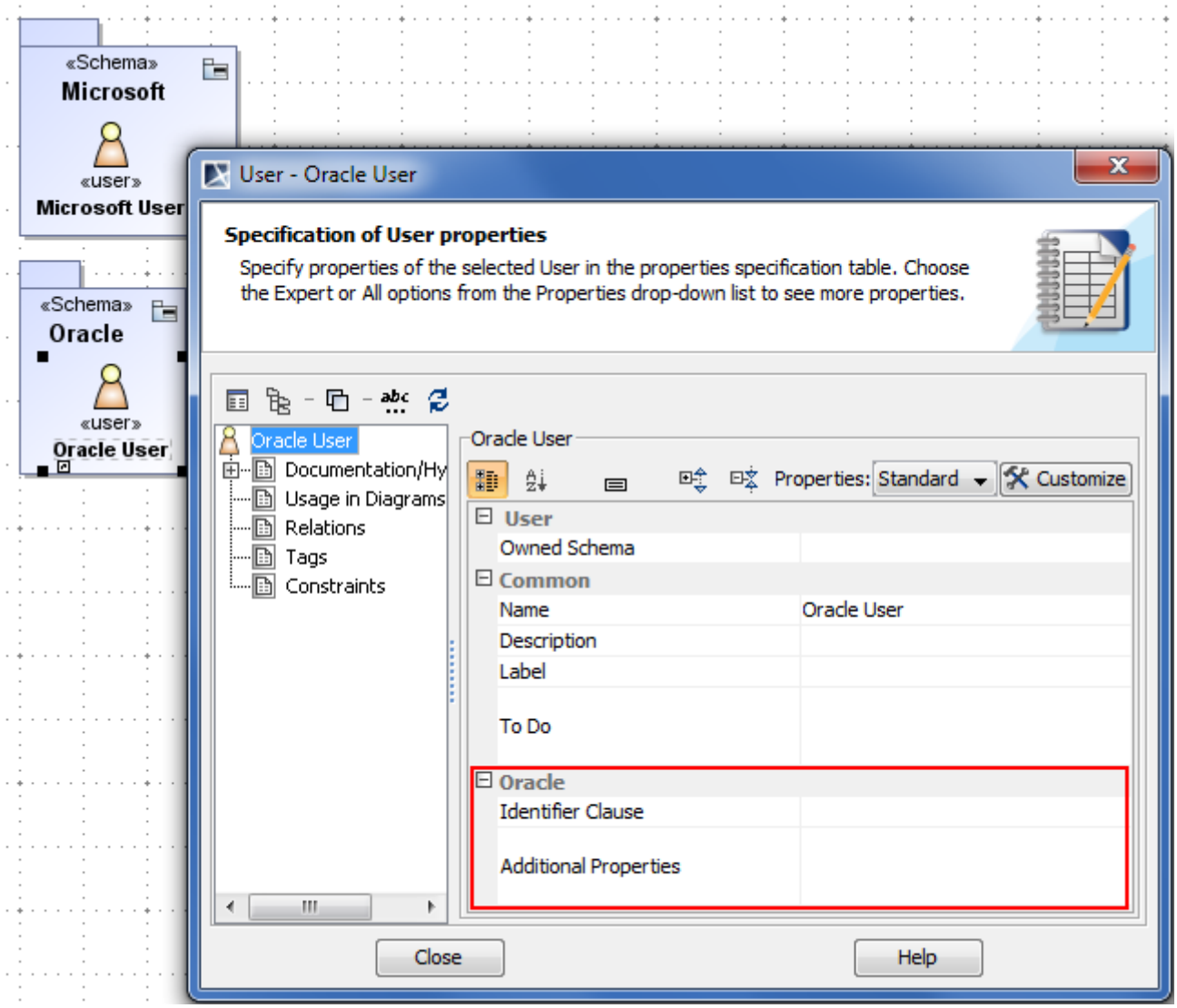


Figure 13 -- Specification window of user that is INSIDE Oracle schema

Meta property substitution (changing name of UML property)

Sometimes standard UML property needs to be customized, to represent more restricted types, or under different name (renamed).

Name substitution

Example:

DSL element should use “blockName” property instead of “name” from UML.

1. Create “name” attribute in ”customization class.
2. Apply <<metaProperty>> stereotype.
3. Specify new name “blockName” in tag “newName” of <<metaProperty>> stereotype.

Type substitution (restriction)

Example:

The type of <<BlockProperty>> shall always be <<Block>>.

1. Create “type” attribute in “customization class”.
2. Apply <<metaProperty>> stereotype.
3. Add <<Block>> stereotype as value of “newTypes” tag.

Suggesting predefined values

There is an ability to create such properties, where list of predefined values will be suggested for the end user (as now with “type modifier”).

1. Create attribute in “customization class”
2. Name attribute as tag which should be customized
3. apply <<metaProperty>> stereotype
4. enter suggested values into “suggestedValues” tag.

Combo box editor with predefined values will be used to edit such customized property.

Attaching element-specific help topics

For each DSL type (the term is defined in the section “Introduction” on page 5 of this user guide), you can define a string value referring to an element-specific help topic, which can be opened for the DSL element in one of the following ways:

- By clicking the **Help** button in the element’s Specification window.
- By pressing F1, when the element’s symbol is selected.

For the information on how to create additional help topics for MagicDraw, please refer to the subsection “Plug-in descriptor” in the chapter “Plug-ins” of “MagicDraw OpenAPI UserGuide.pdf”.

To define a help topic ID for the DSL type

1. Select the appropriate customization class and open its Specification window.
2. Click the **Tags** tab.
3. In the **Tags** specification pane, select the helpID tag and click the **Create Value** button.
4. In the text area on the right, type the help topic ID.
5. Click **Close**.

NOTE	After updating the customization class specification, either the DSL Customization profile must be reused in the model or the model must be reloaded.

In case an element represents two or more DSL types, the first in order DSL type’s help topic ID will be taken.

In case the DSL type customization does not have a helpID tag value defined, the help topic ID will be taken from his ancestor's (more general stereotype) customization helpID tag value. If there is no ancestor, the metaclass help will be opened.

For the detailed information on how to create a customization class, please refer to the section “Creating customization data” on page 22 in this user guide.

Customizing element shortcut menu

The main customization of shortcut menu shall be done using “User perspectives” functionality.

However perspectives just provide ability to hide/show existing menus, but not to create new one.

New shortcut menu items could be created using customization model.

Quick property editor

Some properties are most often used and shall be accessible in shortcut menu of stereotyped elements.

To specify properties that shall be in shortcut menu, use “inShortcutMenu : Property [*]” tag.

Only these types can be edited from the shortcut menu:

- Boolean, boolean – use checkbox editor.
- Enumeration – select one of the listed enumeration literals.
- Reference to one element (sublass of Classifier) – use element list to select type.

Properties of all other types will be ignored.

Quick stereotype applying

Stereotypes as <<continuous>>, <<buffer>> and others act like a flag, so it would be nice to use quick checkbox to set these stereotypes from the element shortcut menu.

To reference metatypes in which shortcut menu quick stereotype applying checkbox shall be added, use tag “quickApplyingFor : Class [*] in customization class.

Example

Customization class of stereotype <<continuous>> shall have tag quickApplyingFor=ObjectNode.

Checkbox editor with stereotype name will be added at the end of shortcut menu for every object node.

Quick element creation from the shortcut menu, customized category

You can group elements with categories in the **New Element** menu in Containment Tree or in various create menus in dialog boxes and specifications. The “category” tag of <<Customization>> stereotype is used for category creation.

Example:

Create the Customization class, specify **category** tag and other tags (see Figure 14 on page 39).

The stereotyped *Business Requirement* class will be added to the package shortcut menu > the **New Element** command > the **Requirements** group (see Figure 15 on page 39).

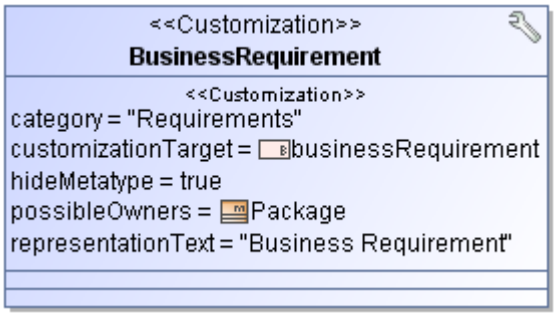


Figure 14 -- The BusinessRequirement class

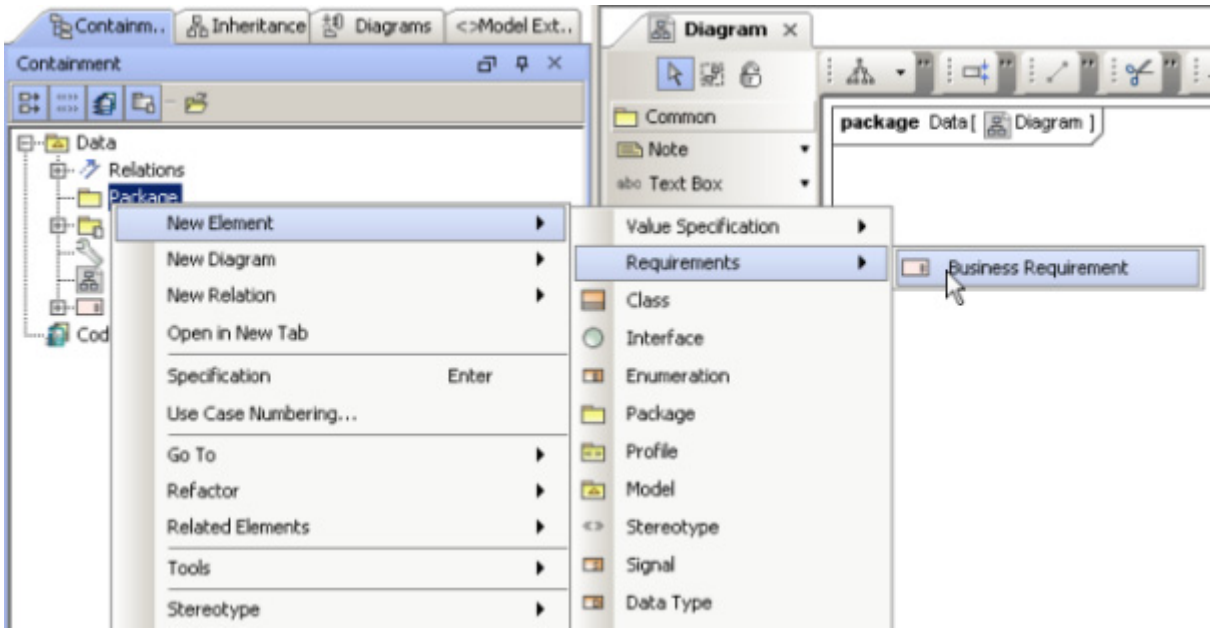


Figure 15 -- In the package shortcut menu, the New Element command, there is added Requirements category with possibility to create stereotyped class Business Requirement

Custom model initialization

DSL customization provides behavior when some model element initialization could be done after applying stereotype on it.

Default values

Tags as any other properties can have default values. These values will be used when tag is created.

Tag definition shall have multiplicity greater then 0, only in this case it will be created automatically when applying a stereotype.

Stereotypes on relationship ends

Sometimes DSL requires applying stereotypes on some elements after stereotyped relationship connection to those elements.

Several tags are defined in <<Customization>> stereotype for this purpose:

- applyToSource : Stereotype [*] – stereotypes that must be applied on the source element of this relationship after connection.
- applyToTarget: Stereotype [*] – stereotypes that must be applied on the target element of this relationship after connection.

“Source” and “Target” are respectively “source” and “target” of directed relationship.

The first connected element will act as “source” for non-directed relationship.

Example:

1. Create stereotype <<serve>> for Dependency.
2. Create stereotypes <<master>> and <<servant>> for Element.
3. Customize <<serve>> stereotype, create customization class and specify tags: .
   - “customizationTarget = <<serve>>
   - “applyToSource = <<master>>”
   - “applyToTarget = <<servant>>”
4. Now, when you will draw <<serve>> dependency from one element to another, stereotypes <<master>> and <<servant>> will be applied on these elements.

Required Generalization or Interface Realization

Sometimes DSL requires that elements should be subtypes of some general abstract class or interface.

Tag for this purpose is defined in <<Customization>> stereotype:

superTypes : Element [*] – types that should be super types of stereotyped element. Generalization or InterfaceRealization (If interface) will be created in the model after such customized stereotype applying.

Example

Every JAVA class should be subclass of Object class, so every Class marked with

<<JAVAClass>> stereotype should be inherited from Object.

- Create customization class for <<JAVAClass>> stereotype
- Specify tag superTypes = Object

Inheritance of DSL customization

For the convenience of end user, DSL customization is inherited when stereotype is inherited, so user don’t need to modify existing DSL customization when extending the (standard) profile.

For example, if user decides to create its own subtype of SysML Requirement, let’s say “Performance Requirement”, it is enough to define new stereotype << Performance Requirement>> and add Generalization between SysML <<Requirement>> and << Performance Requirement>>.

All predefined symbol properties (style) and semantics rules of SysML Requirement will be reused on elements stereotyped by << Performance Requirement>>.

The subtype could have their own customization with some basic tags, like “human name”, etc. Customizations will be merged, so only additions should be defined.

It is duty of DSL customizer to avoid conflicting customizations as result could be unpredictable.

Generalization between <<Customization>> classes itself is also possible.

In this case specific (subclass) customization inherits customization rules if these rules are not redefined in subclass (tag is empty). If both customizations have the same tag filled, the rules are not merged, the one from subclass will be used.

Custom rules for relationships

Stereotyped relationships may have special rules, describing what kind of elements can be connected.

Several tags are defined in <<Customization>> stereotype for that purpose:

- typesForSource: Class[*] – metaclasses or stereotypes that should be allowed to connect as source of this relationship. Types cannot conflict with UML permitted types for this relationship (extended by stereotype - customizationTarget). If relationship is not directed, the first connected end should be as source, second - as target.
- typesForTarget : Class [*] – metaclasses or stereotypes for target.

Subtypes of specified end type are allowed only if type is abstract (like Classifier), otherwise only concrete types must be used.

Example1:

TypesForSource=Classifier – Class, Component and etc will be allowed. Simply stereotyped Classifiers will be allowed also (not DSL type).

Example2:

Class – only Class and simply stereotyped Class is allowed.

Example3:

<<Block>> as type. Only Block shall be allowed, ConstraintBlock is not allowed.

Tags that should be used for customizing non-relationships:

- allowedRelationships : NamedElement [*] – types of relationships that are allowed to connect to this type of element. Only these types shall be allowed if specified.
- disallowedRelationships : NamedElement [*] – types of relationships that are not allowed on this element.

Subtypes of specified allowed/disallowed relationship will be analyzed only when specified relationship type is abstract metaclass (like DirectedRelationship and similar). In other cases only concrete types will be checked.

Example1:

If only <<Trace>> is allowed, subtypes of trace <<copy>> and <<verify>> shall not be allowed.

Example2:

If Association is allowed, Extension (subtype) is not allowed, but simple stereotyped Associations are allowed. All DSL Associations are not allowed.

Example3:

If Dependency is not allowed, simple stereotyped Dependencies like <<delegate>> are not allowed, but subtypes of Dependency like Abstraction and DSL types based on Dependency are allowed.

Customization of possible owned elements

Domain specific diagrams and model requires that some elements of particular predefined DSL types (stereotypes) shall be created inside some other DSL elements.

SysML Example:

Block shall contain BlockProperty elements, but not simple Property.

Block shall own InternalBlockDiagram instead of standard CompositeStructureDiagram

All these cases could be customized using special tags.

Custom owned types

These tags of <<Customization>> stereotype are used to customize owned element types:

- “suggestedOwnedTypes : Stereotype [*]” - contains the list of stereotypes that shall be suggested then inner elements are created in customization target type element.
- “hiddenOwnedTypes : Class[*]” – contains the list of types that shall be not suggested as inner elements. Metaclasses from UML 2,0 metamodel shall be selected. Abstract metaclasses could be selected also (like Classifier, Element); in this case all subtypes will be hidden.

When some stereotyped elements shall be suggested to create in standard UML element, like package, it’s not possible to customize standard (not stereotyped) element.

In this case customization shall be done upside-down, by specifying possible owners.

- “possibleOwners : Class[*]” – all standard UML types, where elements with current customized stereotype shall be suggested to create as owned elements.

Example:

You would like to have possibility to create your own type Project inside standard packages from the browser. Create customization class for <<Project>> stereotype and specify possibleOwners = Package.

Custom owned diagrams

The same principles are used for the inner Diagrams customization:

- suggestedOwnedDiagrams : String[*] – diagram types as Strings. Only these diagrams will be suggested for creation. Other diagrams will be hidden.
- hiddenOwnedDiagrams : String[*] – diagram types as Strings.

Custom suggested relationships

MagicDraw provides the ability to create relationships directly in the model, from model browser, starting from source or target element.

List of suggested relationships could be customized (DSL elements will also be added into this list):

- If DSL relationship has no connection rules for restricted end types and has ”hideMetatype=true” (is DSL type), it will be added into all lists where extended UML type is suggested. (e.g. <<allocation>> in SysML will be suggested everywhere Dependency is suggested)
- If DSL relationship has some connection rules, it will be suggested to create only from/to these restricted types (all connection rules will be analyzed).

Customization of symbols

Custom path style

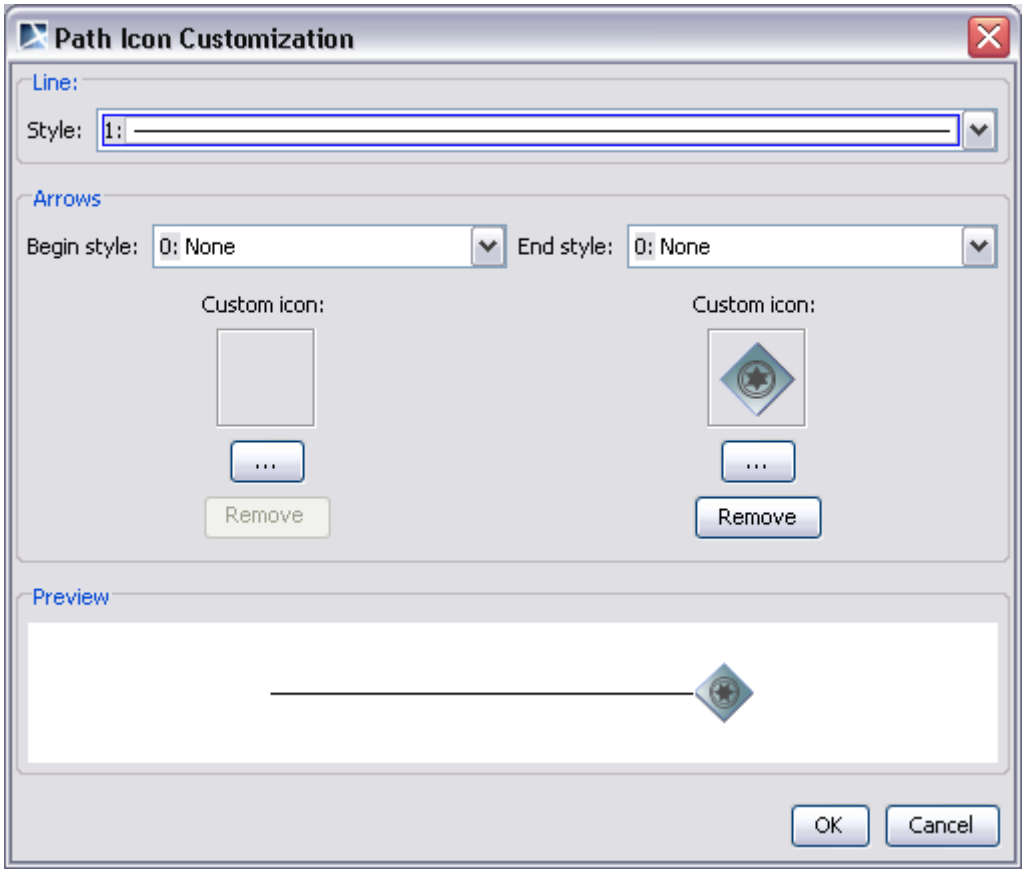
Appearance of standard UML paths is strictly limited by UML specification.

UML provides the ability to customize appearance of stereotyped elements, by specifying stereotypes with icons, but these icons can be effectively used to overlap shapes only, but not paths.

The format of stereotype icon is not restricted by UML, so MagicDraw introduces unique icon format for paths appearance extensions.

There are several properties for paths that could be customized:

- Line style (dashes, dots, solid line, etc).
- Arrow style at first end.Arrow style at second end.
- Custom scalable icon for the first end.
- Custom scalable icon for the second end..



All these properties together forms “path icon” content saved as normal stereotype icon.

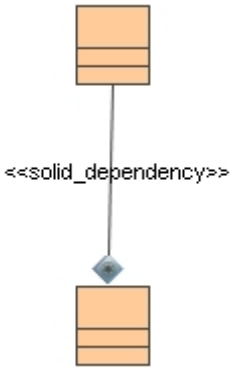
Additionally, “Line width” property is added as standard symbol property for paths.

Stereotype shall extend Relationship (plus Message, Activity Edge, InstanceSpecification, Transition, Connector) in order to use special “path icon”.

Path icon usage rules:

- Custom appearance will be used when stereotype is applied.

- End icons will by rotated according path position and direction
- End icon size will be scaled according font size (the same as for standard arrows)
- Example:



Setting default symbol size of DSL element

Default size can be defined in DSL customization, using new tag in <<Customization>> stereotype (the "Symbol" group):

```
defaultShapeSize : int [2]
```

- The first value is width (x) in pixels. Value= 0 is used if default MD width should remain, but height should be changed.
- The second value is height (y) in pixels.

Default size will be used everywhere when new symbol is created - drag'n'drop, diagram toolbar button, diagram wizards, etc.

Rules of stereotypes that cannot be allowed to apply

When applying stereotype to DSL element, the list of available to apply stereotypes is filtered according to the following rules:

- There is not allowed to apply two DSL customized stereotypes (with hideMetatype=true) to the same element. For example, it shall not show <<Requirement>> stereotype in Block shortcut menu (if <<Block>> is already applied).
- If stereotype should be changed, the current should be unapplied first. (e.g. when changing from <<Requirement>> to <<businessRequirement>>)
- DSL ownership rules are checked and there is not allowed to apply stereotype, which is not allowed in this parent element.
- DSL Relationship rules are checked and there is not allowed to apply some stereotype which source or target element is violating DSL rules for this DSL relationship (the same checking is used while drawing).

Type restriction for custom table in specification dialog

DSL allows creating a new node in specification dialog and showing values of some UML property in the table form.

Since MagicDraw 16.5 version, it is possible to filter elements according type:

- New tag was added into <<PropertyGroup>> stereotype used for DSL

- filter : Type [*] - types could be UML metaclasses and custom stereotypes.

DSL engine checks this value and displays only elements of these types in the table.

- The "Create" button allows creating a new elements of these restricted types only.

Rules for subtypes

- If type is stereotype, all subtypes (sub stereotypes) are accepted also (e.g. if type is Requirement, BusinessRequirement is accepted also).
- If type is abstract metaclass, all subclasses are accepted (e.g. if type is Classifier - Class, Component and other is accepted). Stereotyped elements are accepted also.

Hiding DSL elements in the type selection dialog

A new property has been added in the DSL Customization dialog that allows you to exclude elements from being suggested as a type in the list dialogs. Select the *doNotSuggestAsType : Boolean* property in the **Customization** dialog, under the **General** group to exclude an element (Figure 16 on page 45).

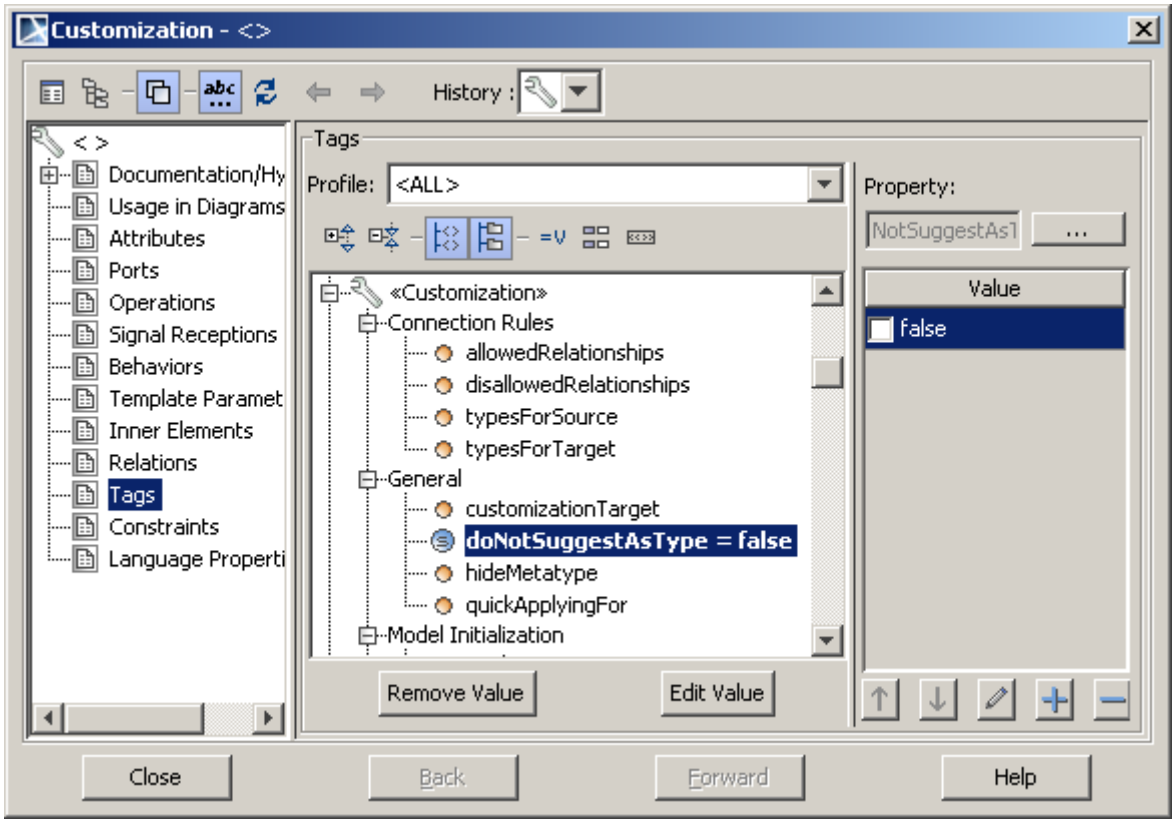


Figure 16 -- The Customization Specification Dialog, Tags branch

Extending metamodel with derived properties

As of MagicDraw version 16.8, DSL customization allows for extending the UML metamodel or its profile (such as SysML or UPDM) by creating new read-only derived properties.

A derived property is the one, whose values are calculated automatically from the other properties' values.

It is important to notice that the derived properties are calculated on the fly. MagicDraw analyzes the derived properties and calculates them for the existing model elements, when loading a model, and dynamically updates them according to the model changes.

The derived properties can be added either to the standard UML elements, or to the ones that are extended with stereotypes. In order to extend the metamodel with the derived properties, there is no need to apply the stereotypes - the metamodel is extended in runtime.

You can define the derived properties in different expression types: Simple, Meta Chain, OCL, and Binary (reference to java code).

You can easily access derived properties of a select model element, navigate to their specifications, or use them in the model analysis tools, such as dependency matrices, report templates, an relation maps.

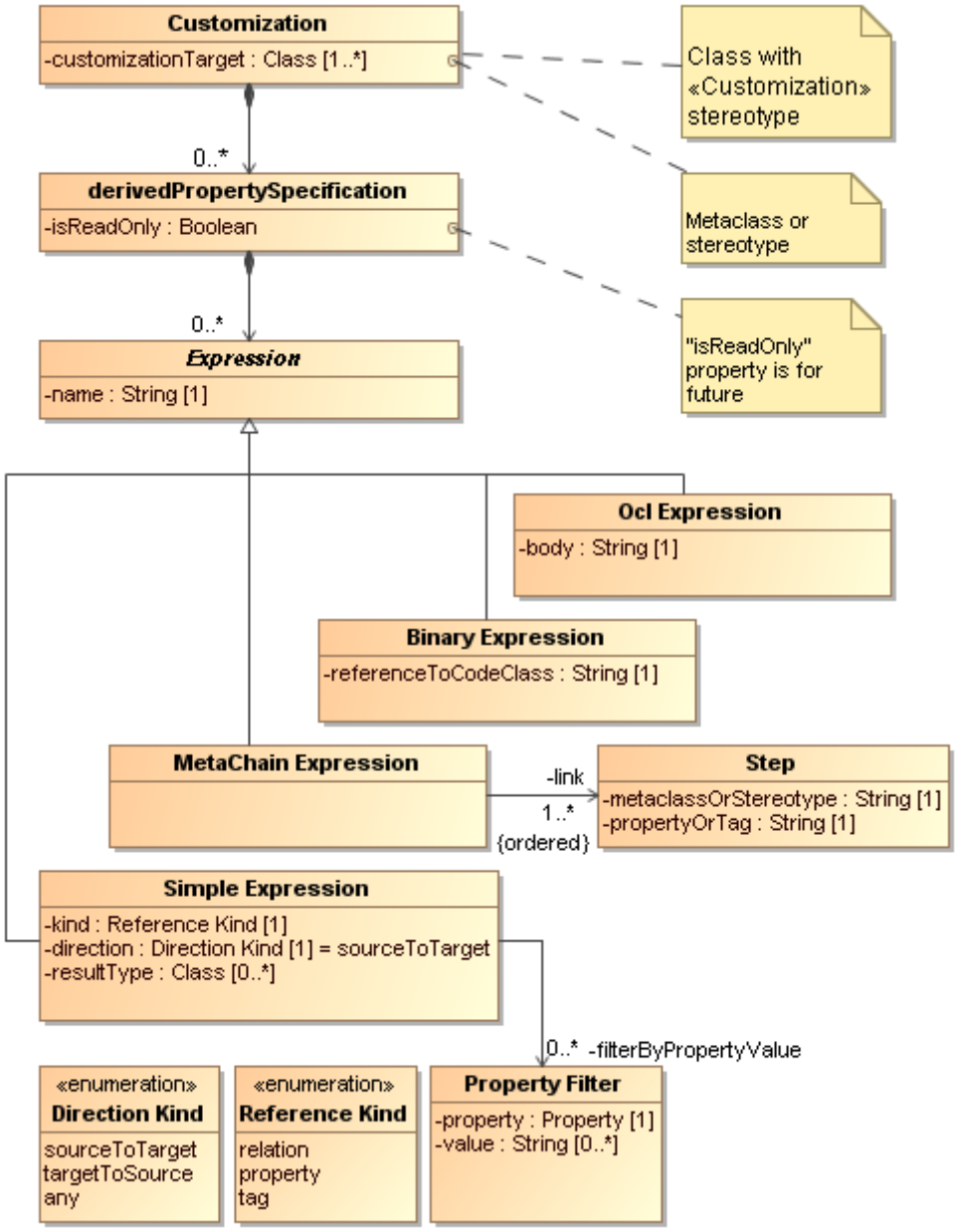


Figure 17 -- Derived properties specification metamodel

Specifying derived properties

In order to be included in a MagicDraw project as a property extending a UML element, the property (attribute) must be owned by a DSL customization class and must be stereotyped as <<derivedPropertySpecification>>.

To create and specify a derived property

1. Select the customization class, for which you want to create a derived property.

IMPORTANT!	The derived property will be created for the stereotype or metaclass, which is defined as a customization target in the selected customization class specification.

2. From it's shortcut menu select **New Element** > **Property**.
3. Type the property name.

NOTE	It is recommended to type derived property names in camel case, e.g., "realizingActivities". A property name, that has been defined in camel case, will be displayed in separate words with the first word capitalized, e.g., "realizingActivities" will be changed to "Realizing Activities" for showing in the user interface.

4. Apply the stereotype <<derivedPropertySpecification>> to this property. The result view of the Model Browser must be appropriate to the picture below.

Figure 18 -- An example of the customization class with the attribute for the derived property specification

5. In the property's Specification window, edit property values specifying the derived property.

NOTE	All the properties are specified in the general specification pane. Some of them can also be specified in the **Tags** specification pane. For detailed information, see the table below.

IMPORTANT!	If you do not see the newly created derived property in the element's Specification window, read the following information. It may help you. • In the element's Specification window, the specified derived property is visible by default only in the **All** properties mode. So first of all try to swith to this mode. Furthermore, you can change the mode, in which you want the derived property to be visible (click the **Customize** button). • If you have specified the derived property for the element, which has the <<usedUMLProperties>> tag of DSL customization specified, you must add the newly created derived property to the element's visible properties list. For the detailed information, please refer to the section "To make derived properties of customized elements which has usedUMLProperties tag specified visible" on page 69.

The properties that are specified for the derived property are described in the table below.

Property	Description	Specified in...
Name	The name of the property. This name will be added and visible in the specification of the element, whose type is defined as a customization target.	General specification pane.

Property	Description	Specified in...
Type	The type of the derived property values. You may set a model element type or a data type (e.g., string, boolean, or integer from the UML Standard Profile) as a property value. Example: If you want classes and components to be displayed as property values, select the classifier as a type for the derived property value.	General specification pane.
Multiplicity	The multiplicity of the derived property values. Example: If the multiplicity is “0..1”, the derived property value will be shown as single. If the multiplicity is “0..*”, the derived property values will be shown as collection.	General specification pane.
Is Read Only/ isReadOnly	If true, the derived property is read-only, meaning that it is not allowed to edit the result elements for the derived property value. If false, the property is read-write, meaning that it is allowed to edit the result elements for the derived property according to the expressions defined. NOTE: Currently only the read-only derived properties are supported.	• General specification pane. • **Tags** specification pane.
Is Ordered	If true, the property values are always displayed in the same order. If false, the property values each time are displayed in non-predictable order.	General specification pane.
Is Unique	If true, the property values are unique, i.e., the same element is displayed only once. If false, the property values may be displayed more than once.	General specification pane.
Expression/ expression	One or many expressions of the derived property. An expression defines the criterion for selecting the result elements. There are four possible expression types: • Simple (UML relationships, properties, and tags) • Multi properties chain (Meta Chain) • OCL • Binary (Reference to code class) The criterion for selecting the result elements can also be any combination as union of the expression types above. For the detailed information, see “Defining expressions” on page 50.	• General specification pane. • **Tags** specification pane.
Documentation	The text that will be displayed as the derived property description in the element’s Specification window and **Properties** panel.	**Documentation** specification pane.
customizationTarget	One or many element types (stereotypes and/ or metaclasses), for which the specified derived property will be added. **NOTE:** The element type of the customization target is the one, from which the expression calculations are started.	Customization class Specification window > **Tags** specification pane.

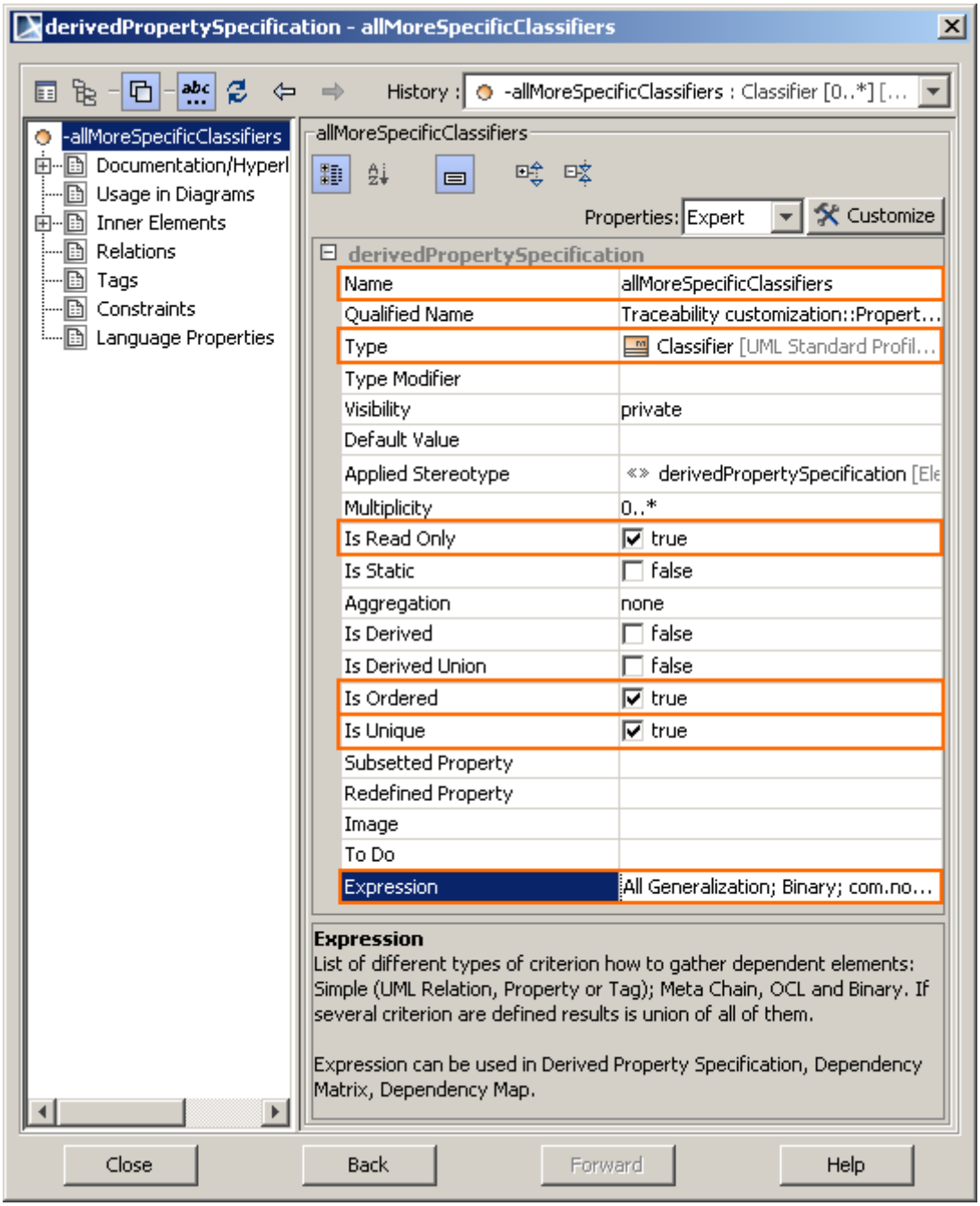


Figure 19 -- Derived property specification. General specification pane

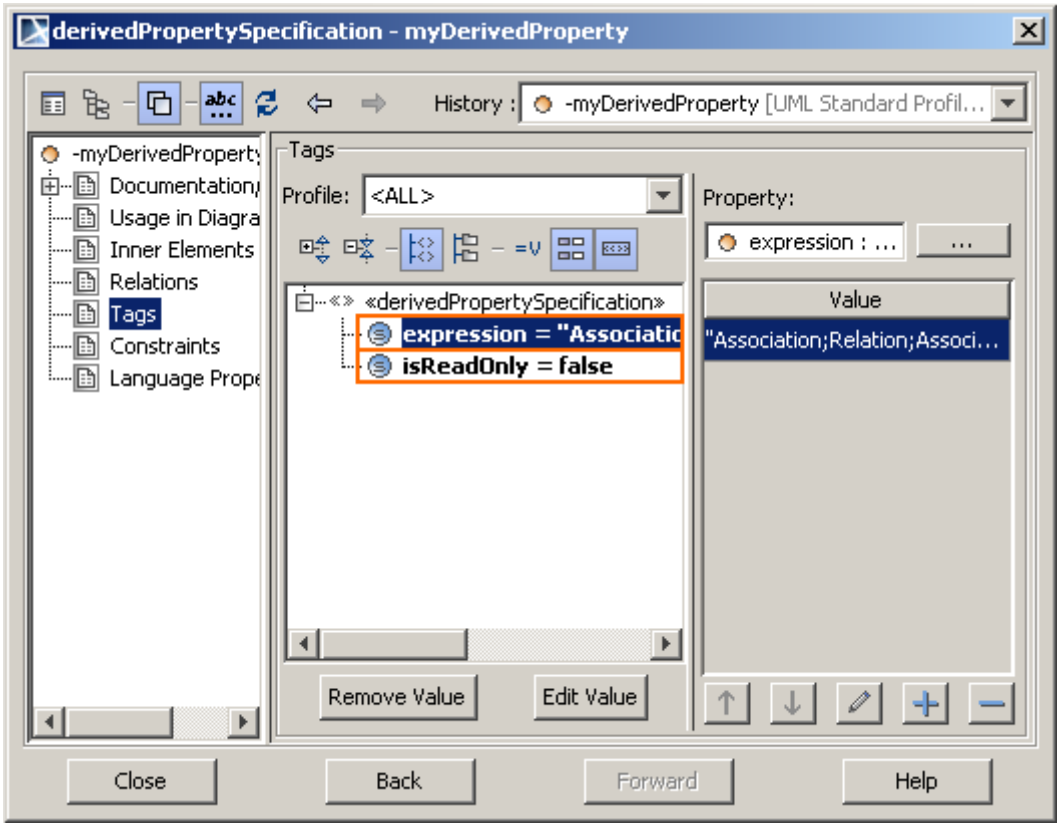


Figure 20 -- Derived property specification. Tags specification pane

Defining expressions

Each derived property must have one or more expressions defining the criteria according to which the result elements are searched.

To open the dialog for a new expression definition

IMPORTANT!	It is strongly recommended to have the customization target specified in the customization class before defining the expression for its derived property.

1. Select the property with the stereotype <<derivedPropertySpecification>>.
2. Open the property's Specification window.
3. Do either:
   - In the general specification pane (opened by default), select the **Expression** property and in the property value cell click the "..." button.
   - Click on the **Tags** tab and in the **Tags** specification pane, select the "expression" tag and then click the **Create Value** button.

As the **Criterion Editor** dialog opens, you are ready to define one or more criteria for calculating the derived property values.

NOTE	If there is at least one expression already created, you can open the **Criterion Editor** dialog by clicking the Edit or Add buttons at the lower right corner of the **Tags** specification pane.

The dialog contains four tabs, each dedicated for one expression type:

- **Simple** (for detailed information, see section “Simple expressions” on page 51)
- **OCL** (for detailed information, see section “OCL expressions” on page 55)
- **Reference to code class** (for detailed information, see section “Binary expressions” on page 60)
- **Meta Chain** (for detailed information, see section “Meta Chain expressions” on page 63)

Simple expressions

The simple expressions editor depicted in the figure below allows expressing direct dependencies between the elements through the custom UML relations, properties, and tags, with some extra options to select the result element type, link direction, and additional filters (for relations only).

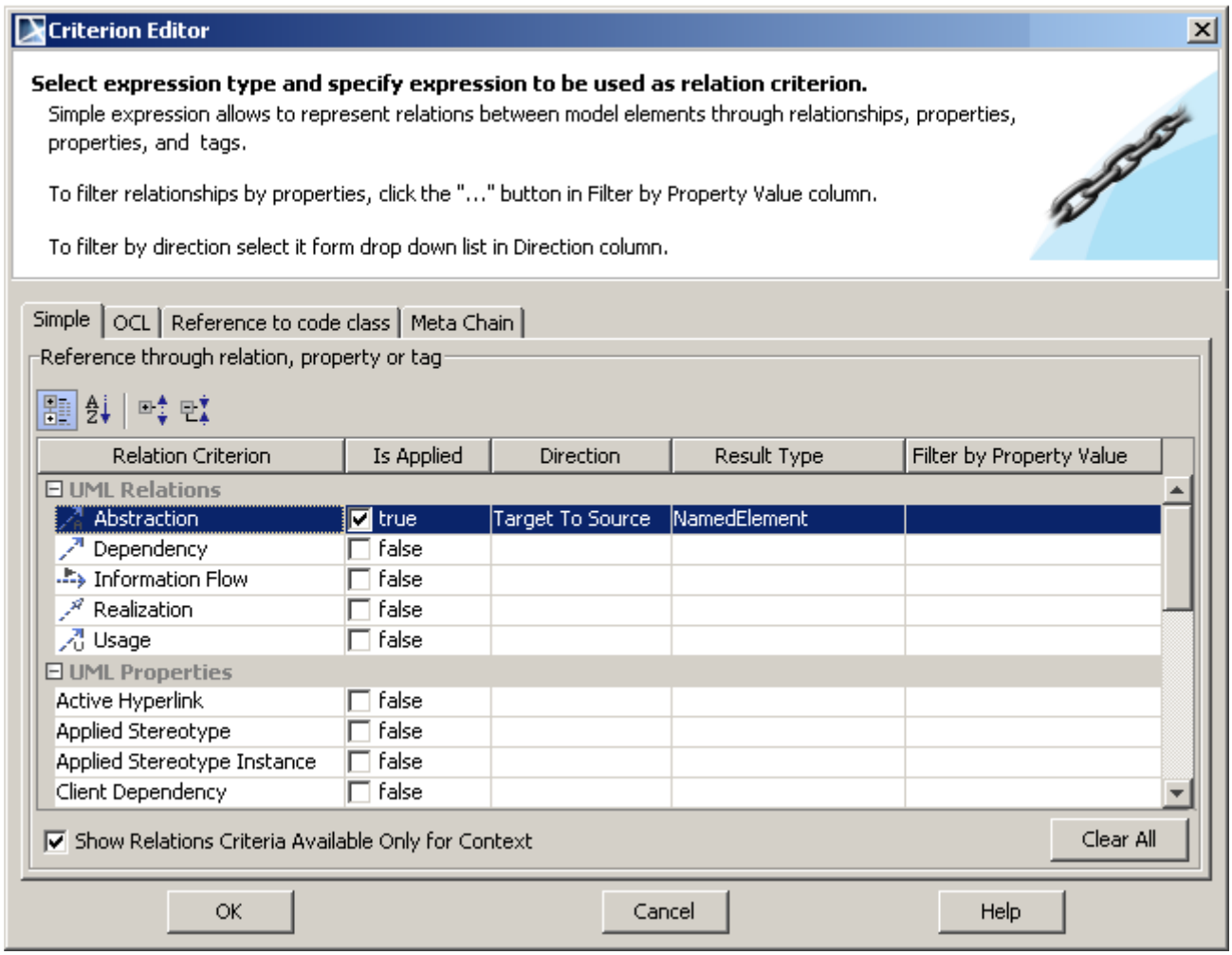


Figure 21 -- Criterion Editor dialog for defining simple expressions

Column	Description
Relation Criterion	A list of UML relations, properties and tags. To manipulate the values display mode in the list, you can use the **Show Relations Criteria Available Only for Context** checkbox.
Is Applied	If selected, an appropriate relation criterion will be applied as an expression for a derived property. Multiple selection is allowed. To clear the selections in this column, click the **Clear All** button. **NOTE:** Clicking this button will not clear the other settings (direction, result type, and filter by property) for an appropriate relation criterion.

Column	Description
Direction	A direction for the expression analysis applied for relations, properties, and tags. Select one of the following values: • Source To Target (default) • Target To Source • Any A source is an element defined as a customization target in the customization class. **NOTE:** The element type of the customization target is the one, from which the expression calculations are started. A target is an element, which is a result of the defined expression. If the direction is defined for relations: • The **Source To Target** direction means that only the outgoing relations that are pointing from the source element to the target element will be treated as a result of this criterion. • The **Target To Source** direction means that only the incoming relations that are pointing from the target element to the source element will be treated as a result of this criterion. • If the **Any** direction is chosen, the both above described cases will be treated as a valid result. If the direction is defined for properties or tags: • The **Source To Target** direction means that only the properties that exist in the source element will be treated as a result of this criterion. • The **Target To Source** direction means that only the properties that exist in the target element will be treated as a result of this criterion. • If the **Any** direction is chosen, the both above described cases will be treated as a valid result.
Result Type	A result elements type compatible and usually specifying a derived property values type, for example, a component, specifying a classifier, which is set as a derived property values type. It is allowed to select more than one result type for a relation criterion. To open the dialog for selecting the result element types, click the "..." button.
Filter by Property	A property of the selected relation criterion used as a more specific filter for selecting derived property values. It is allowed to select more than one property. To open the dialog for selecting the properties as filters, click the "..." button, which is available for relations criteria only. NOTE: Only primitive enumeration properties and the Applies Stereotype property can be used as more specific filters for criteria.

The description of the check box in the **Criterion Editor** dialog for the simple expressions is as follows:

Show Relations Criteria Available Only for Context	If selected, there will be shown only these relation criteria that are available for the UML element, which is set as a customization target. If cleared, all the properties' tags and relationships will be shown as possible criteria.

Example:

Let's say we have a customization class with the use case element type set as a customization target. We will create a derived property for this customization class, name it "realizingClass", and add an expression searching for the classes that are related to the extended use case directly by the public abstraction relationship in the target to source direction (see Figure 22 on page 53).

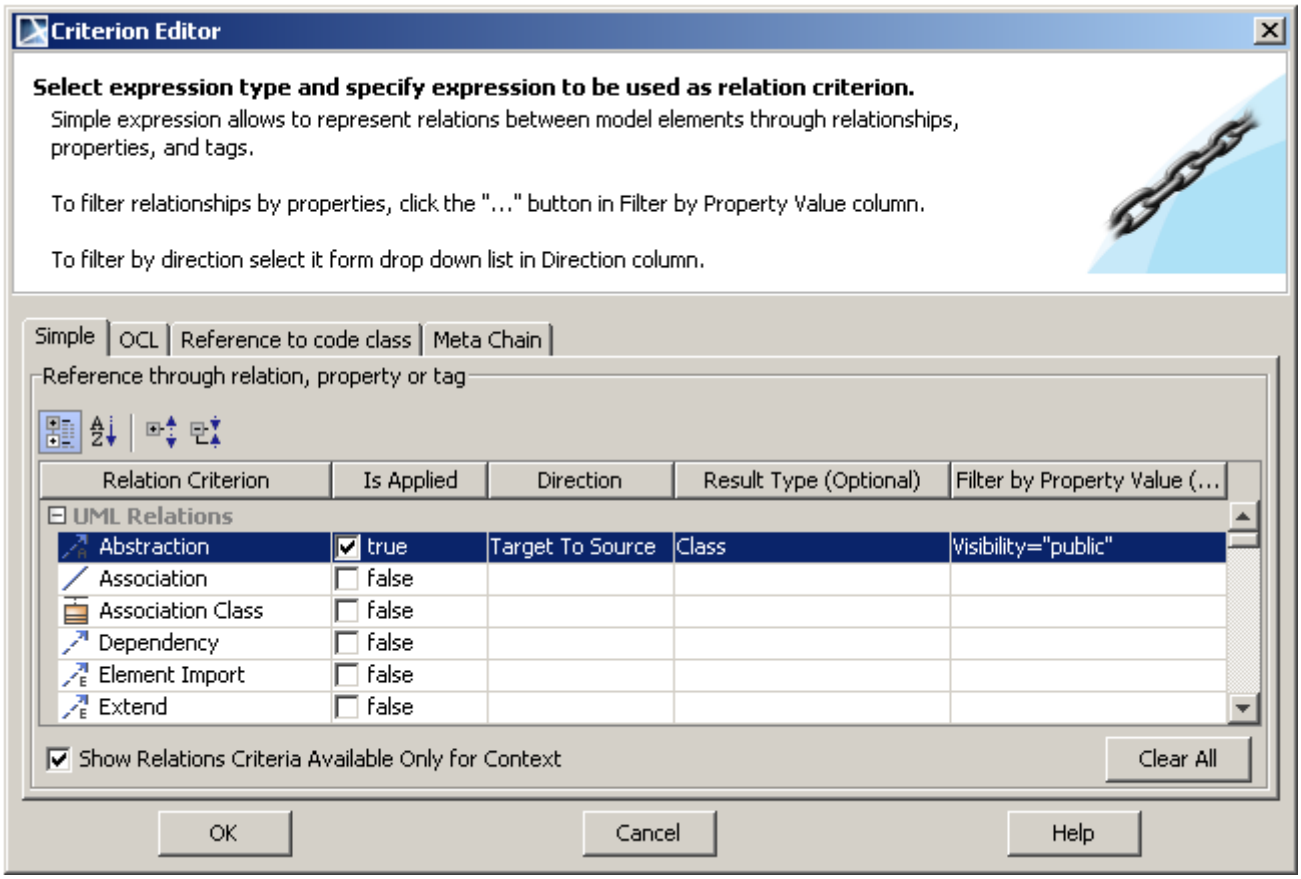


Figure 22 -- A simple expression for the Realizing Class derived property

Thereafter, we will create a class diagram and two elements: a use case, named "Create user", and a design class, named "UserDetails", related by an abstraction relationship as it is show in Figure 23 on page 54.

Now you can see the **Realizing Class** property in the Specification window of the "Create user" use case. The property value points to the "UserDetails" class.

Note that Figure 23 on page 54 shows the **Realizing Class** property in the **Traceability** tab under the **Realization** group. As this example does not show how to create property groups and subgroups, and assign to them derived properties, please, refer to the following sections:

- "Creating your own property groups and subgroups" on page 24.
- "Derived properties visibility" on page 69.

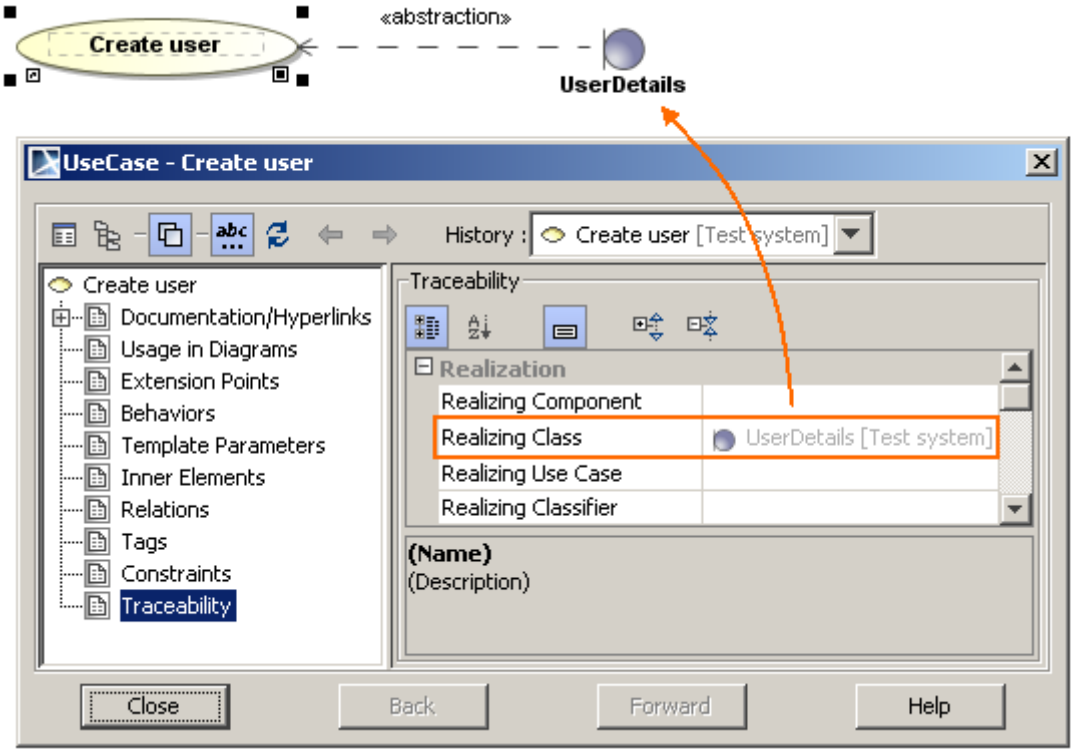


Figure 23 -- A Realizing Class derived property value in the extended element's Specification window

OCL expressions

The OCL expressions editor depicted in the figure below allows for defining an OCL expression for gathering the collections of the result elements.

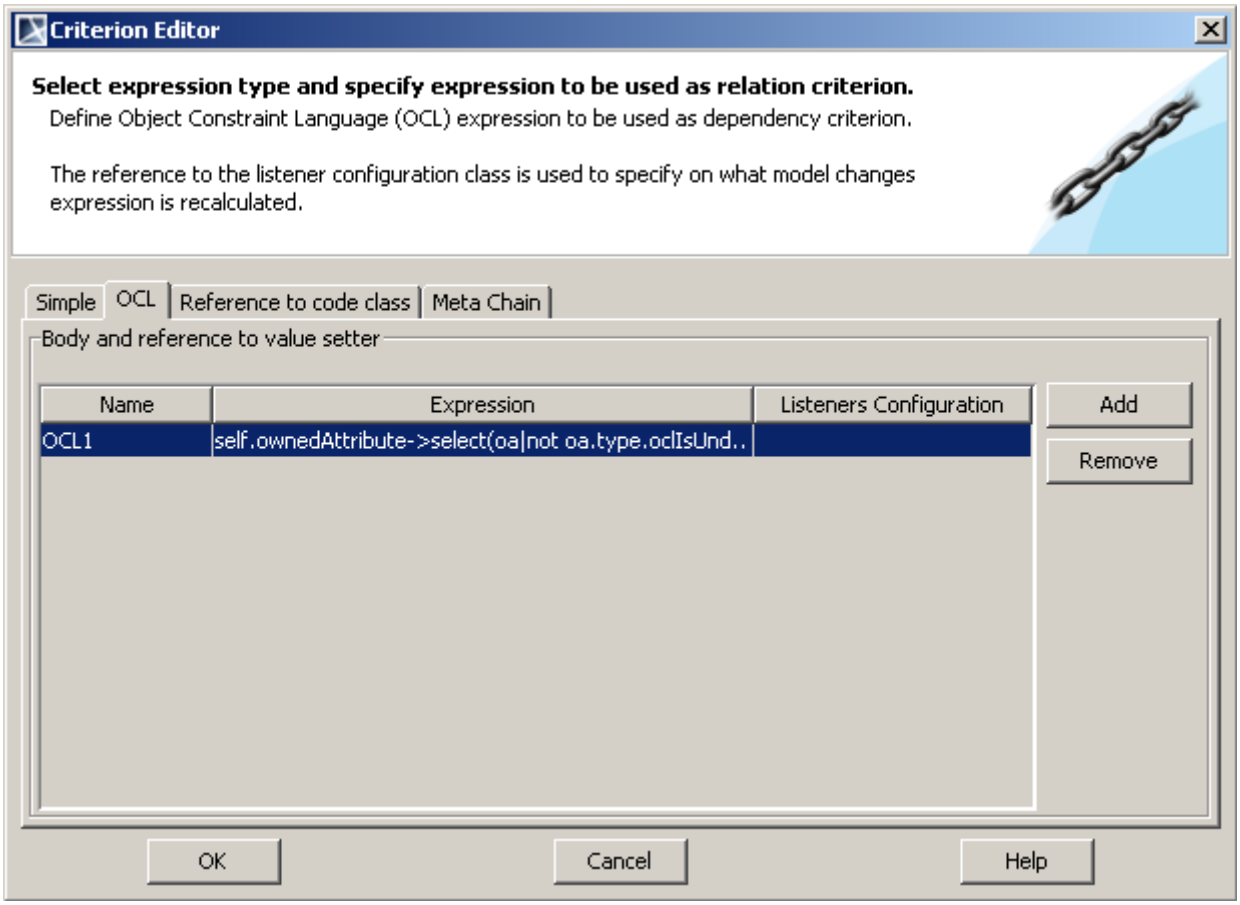


Figure 24 -- Criterion Editor dialog for defining OCL expressions

Column	Description
Name	A name of an OCL expression. It is mandatory and has a default value “Untitledn”, where “n” stands for the expression sequence number. You can edit the name either directly in a cell, or in the **Name** dialog, opened by clicking the “...” button in this cell. To create an expression, click **Add**. To remove an expression, click **Remove**.
Expression	An OCL expression. You can edit the expression either directly in a cell, or in the **OCL Expression** dialog, opened by clicking the “...” button in this cell. **NOTE:** You can check the OCL syntax of your expression, when writing the expression in the **OCL Expression** dialog. To enable the OCL syntax checking mode in the dialog, select the **Check OCL syntax** box.
Listeners Configuration	A reference to a java class, the so-called diagram event listener, which specifies when the OCL expression results must be recalculated. **NOTE:** For the instructions on how to create your own listener using MagicDraw API, please, refer to the chapter “Diagram Events” in “MagicDraw OpenAPI UserGuide.pdf”.

Example:

Let's say we have a customization class with the class element type set as a customization target. We will create a derived property for this customization class and name it "ownedPropertyTypes". Now we add an OCL expression, which searches for the owned attribute types (see Figure 25 on page 56) and recalculates them every time a new attribute is added to the class (see Figure 26 on page 57).

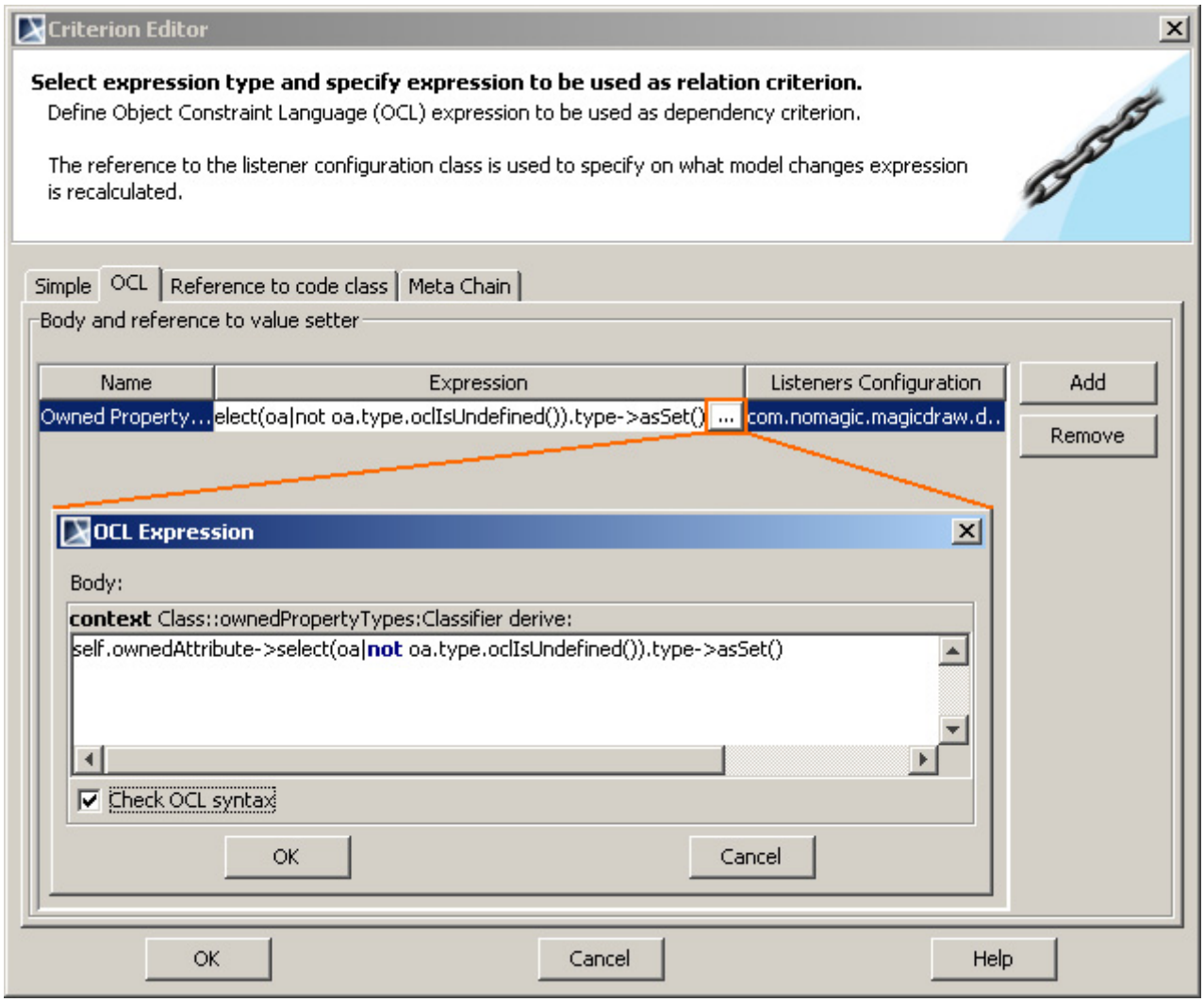


Figure 25 -- An OCL expression for the Owned Property Types derived property

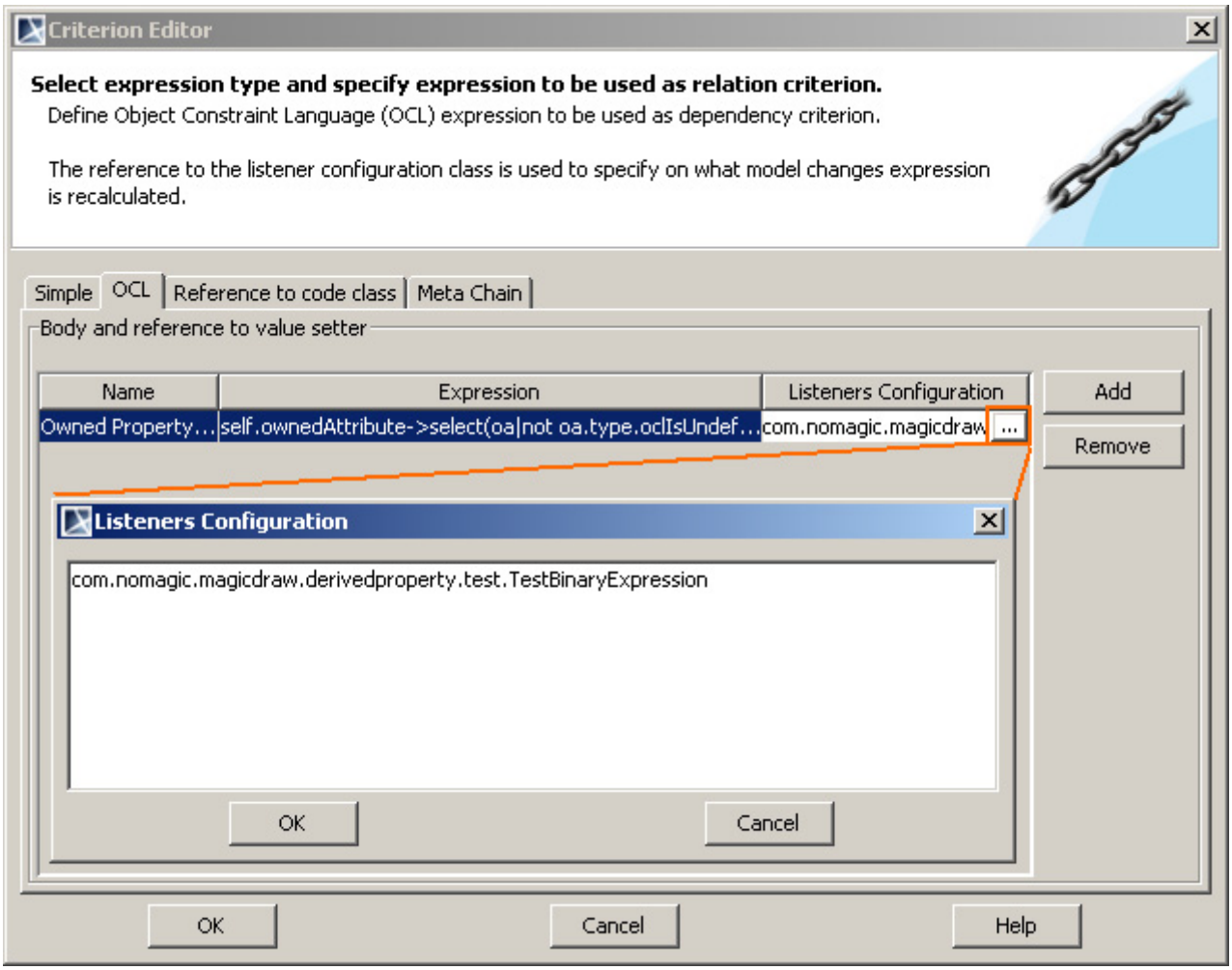


Figure 26 -- A reference to the event listener in the OCL expression

Thereafter, we will create a class diagram and a class named “Person”. Then we will add in this class these two attributes: “ID” and “name”. As it is depicted in Figure 27 on page 58, the **Owned Property Types** derived property in the class Specification window shows two types of the appropriate attributes owned by the class.

After a new attribute is added into the class, the **Owned Property Types** derived property values are recalculated. For the recalculation result, see Figure 28 on page 59.

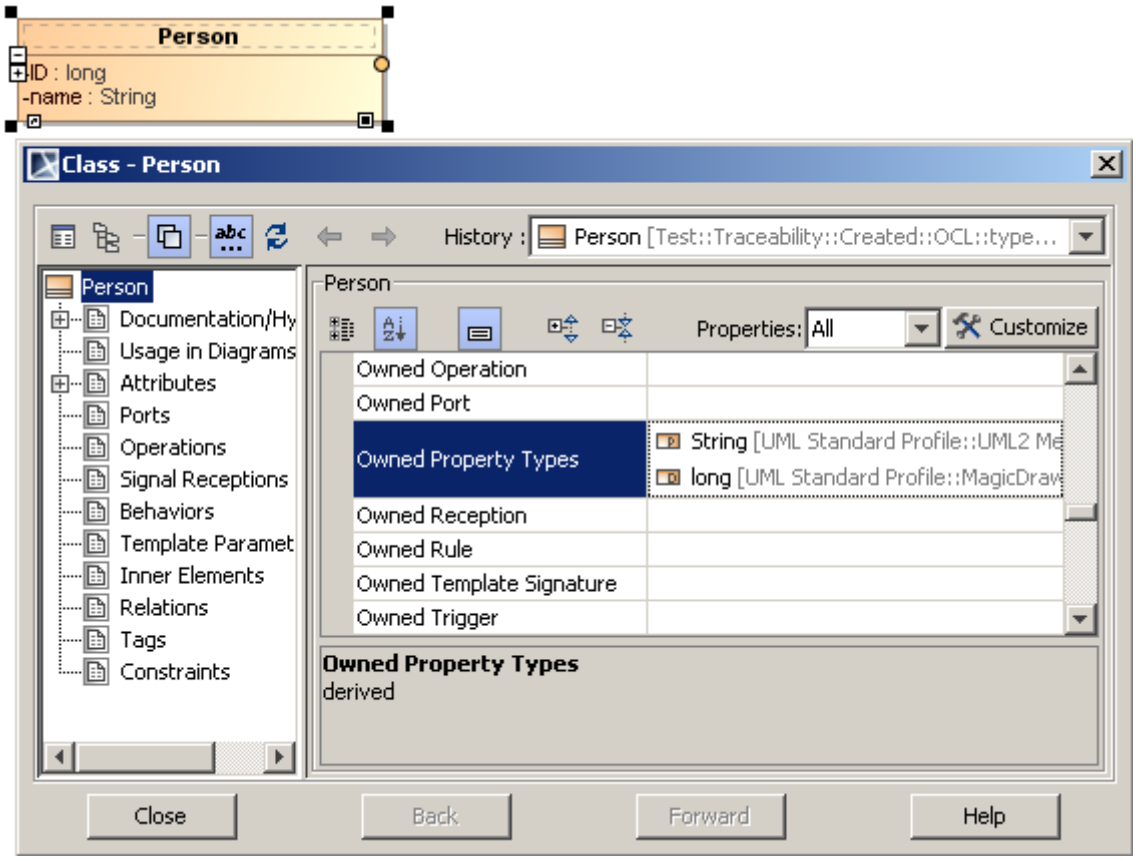


Figure 27 -- Values of the Owned Property Types property before the update of the "Person" class

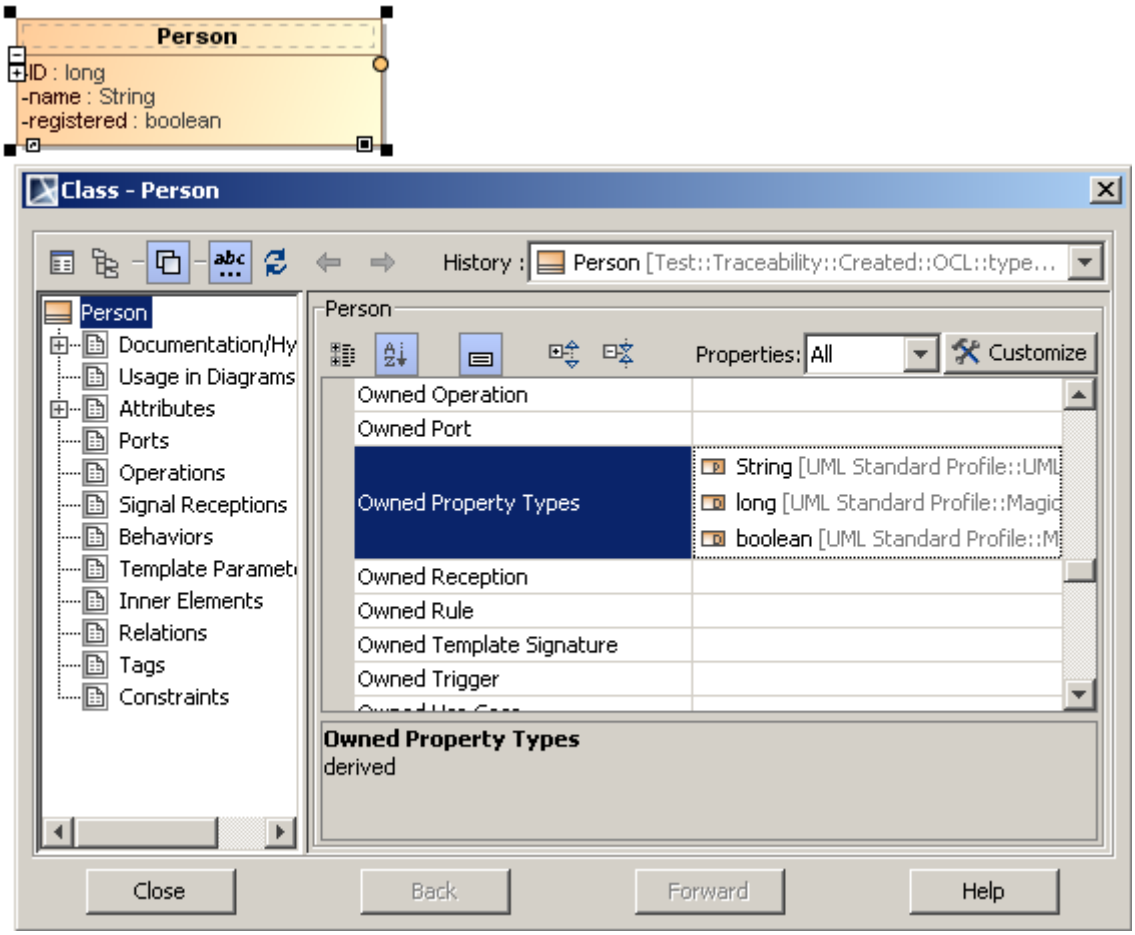


Figure 28 -- Values of the Owned Property Types property after the update of the "Person" class

Binary expressions

The binary expressions editor depicted in the figure below allows for defining a string reference to a java class, which searches for the result elements according to the given parameters.

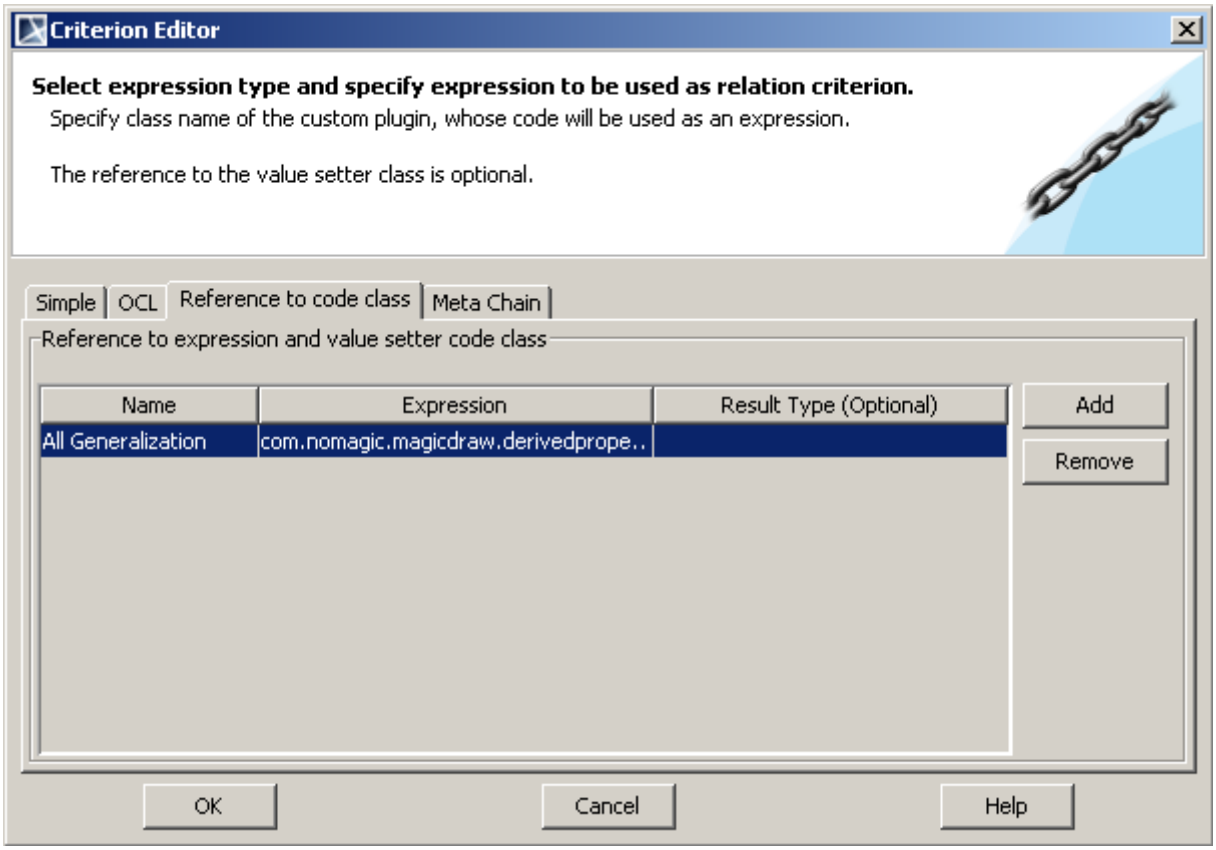


Figure 29 -- Criterion Editor dialog for defining binary expressions

Column	Description
Name	A name of a binary expression. It is mandatory and has a default value “Untitledn”, where “n” stands for the expression sequence number. You can edit the name either directly in a cell, or in the **Name** dialog, opened by clicking the “...” button in this cell. To create an expression, click **Add**. To remove an expression, click **Remove**.
Expression	The name of the java class, which calculates the property value. The specified java class must implement interfaces that are described in the section “How to implement a binary expression” on page 60.
Result Type	A result elements type specifying a derived property values type, for example, a component. It is allowed to select more than one result type for a criterion. To open the dialog for selecting the result element types, click the “...” button.

How to implement a binary expression

A binary expression class should implement the following interfaces:

- `com.nomagic.uml2.ext.jmi.reflect.Expression`

or

- `com.nomagic.magicdraw.derivedproperty.ParametrizedExpression`

and

- `com.nomagic.magicdraw.validation.SmartListenerConfigurationProvider`

An implementation class should have the default constructor. This constructor will be used to create an instance of the class.

If you **do not want to pass any parameter** to your binary expression after the initialization, then it is required to implement the `com.nomagic.uml2.ext.jmi.reflect.Expression` interface.

If you **want to pass parameters** to your binary expression after the initialization, then you have to implement the `com.nomagic.magicdraw.derivedproperty.ParametrizedExpression` interface.

For example, `com.nomagic.magicdraw.derivedproperty.RecursiveExpression` class implements the `ParametrizedExpression` interface. You can specify parameters for the `ParametrizedExpression` interface in the appropriate cell of the **Expression** column using the following syntax:

```
YourParameterizedExpression(param1, param2, ..., paramN)
```

An implementation of the `com.nomagic.magicdraw.validation.SmartListenerConfigurationProvider` interface should return the configuration that identifies, which properties are importantto the expression. The expression will be recalculated, in case one of these properties changes.

TIP!	You are welcome to study the sample binary expression named `AttributeTypesExpression.java`, which is located in <MagicDraw installation directory>\openapi\examples\expression. This sample binary expression is intended to collect types of attributes owned by a class.

How to make the binary expression class available

Create a MagicDraw plugin. The initialization of the plugin should register the plugin's class loader.

TIP!	Please, refer to the expression named `ExpressionExamplePlugin.java`, which is located in <MagicDraw installation directory>\openapi\examples\expression.

Example:

As an example of the binary expression, we will review the recursive derived property, named “allSpecificClassifiers”. This derived property is specified in the Traceability customization profile for the

customization class with the classifier set as a customization target. The binary expression of this derived property recursively searches for the elements that specify the selected classifier (see Figure 30 on page 62).

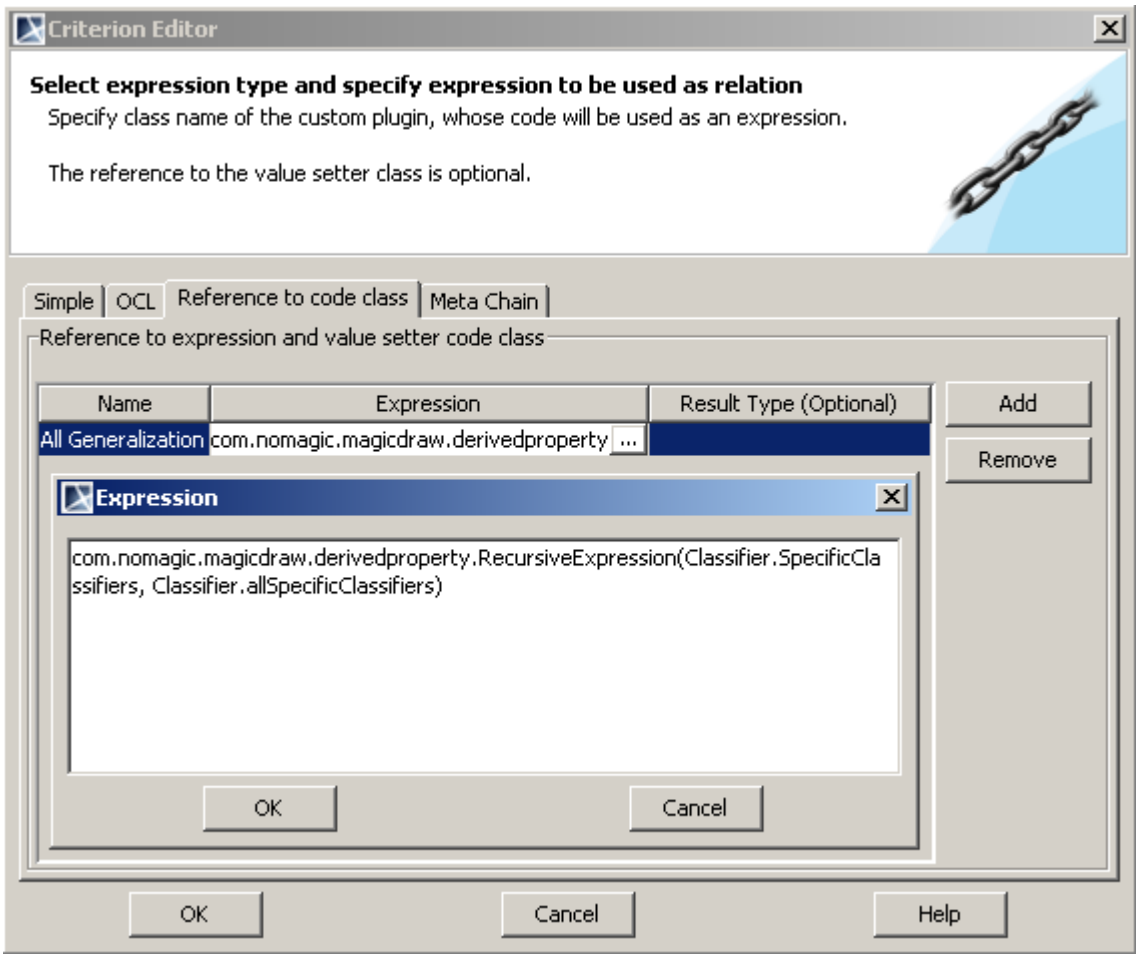


Figure 30 -- An binary expression for the All Specific Classifiers derived property

Let's create a class diagram and five classes related by generalization relationships as it is depicted in Figure 31 on page 63. Then open the Specification window of the "User" class to view the **All Specific Classifiers** property values. The values point to the classes that are both directly and indirectly related to the "User" class (see Figure 31 on page 63).

Note that Figure 31 on page 63 shows the **All Specific Classifiers** property in the **Traceability** tab under the **Other** group. As this example does not show how to create property groups and subgroups, and assign to them derived properties, please, refer to the following sections:

- "Creating your own property groups and subgroups" on page 24.
- "Derived properties visibility" on page 69.

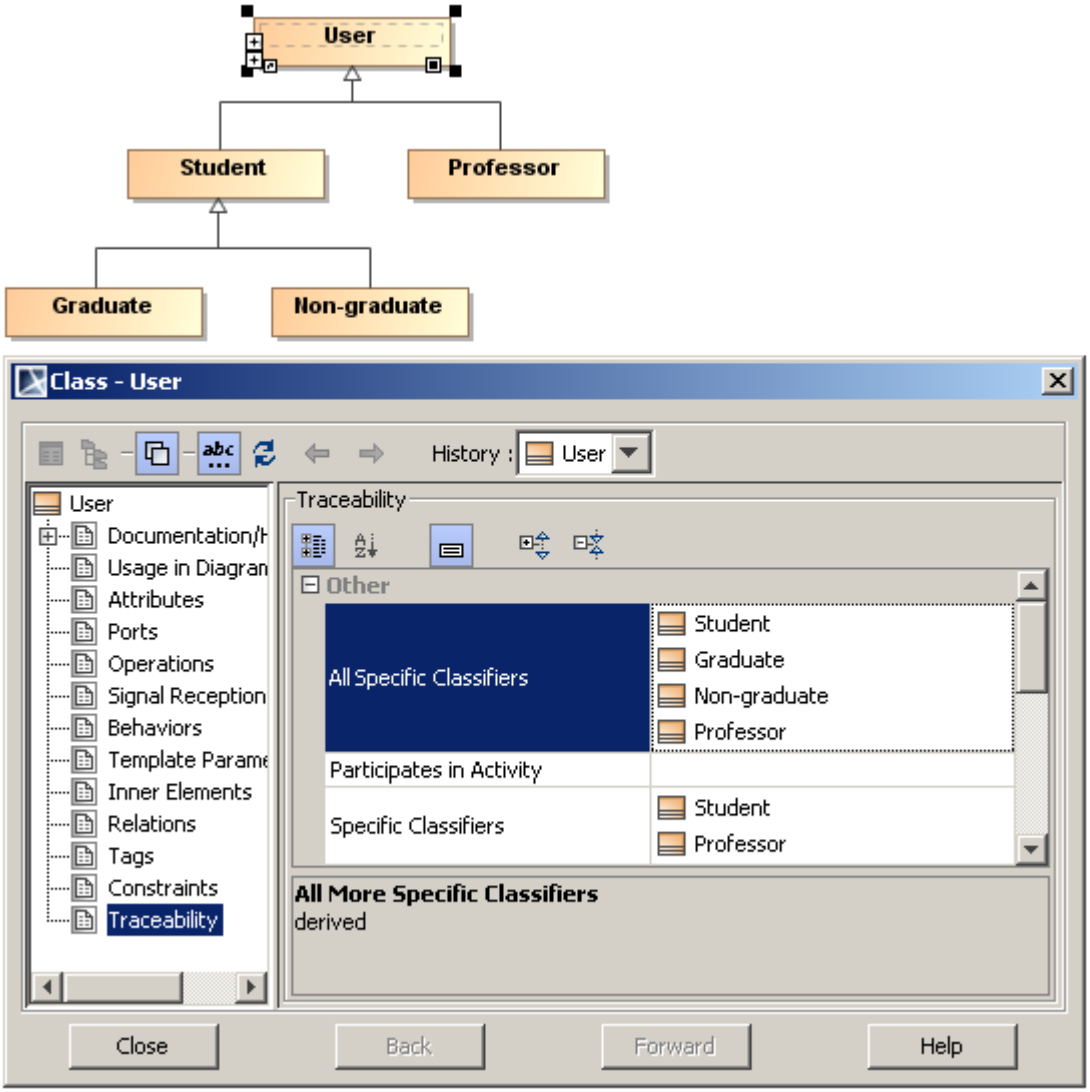


Figure 31 -- Values of the All Specific Classifiers property

Meta Chain expressions

The meta chain expressions editor depicted in the figure below allows for defining a multi properties chain navigating from a context element to a final link property for gathering the result elements as derived property values.

IMPORTANT!	Though the meta chain expressions allow searching for the indirectly related elements, they do not support loops and recursions.

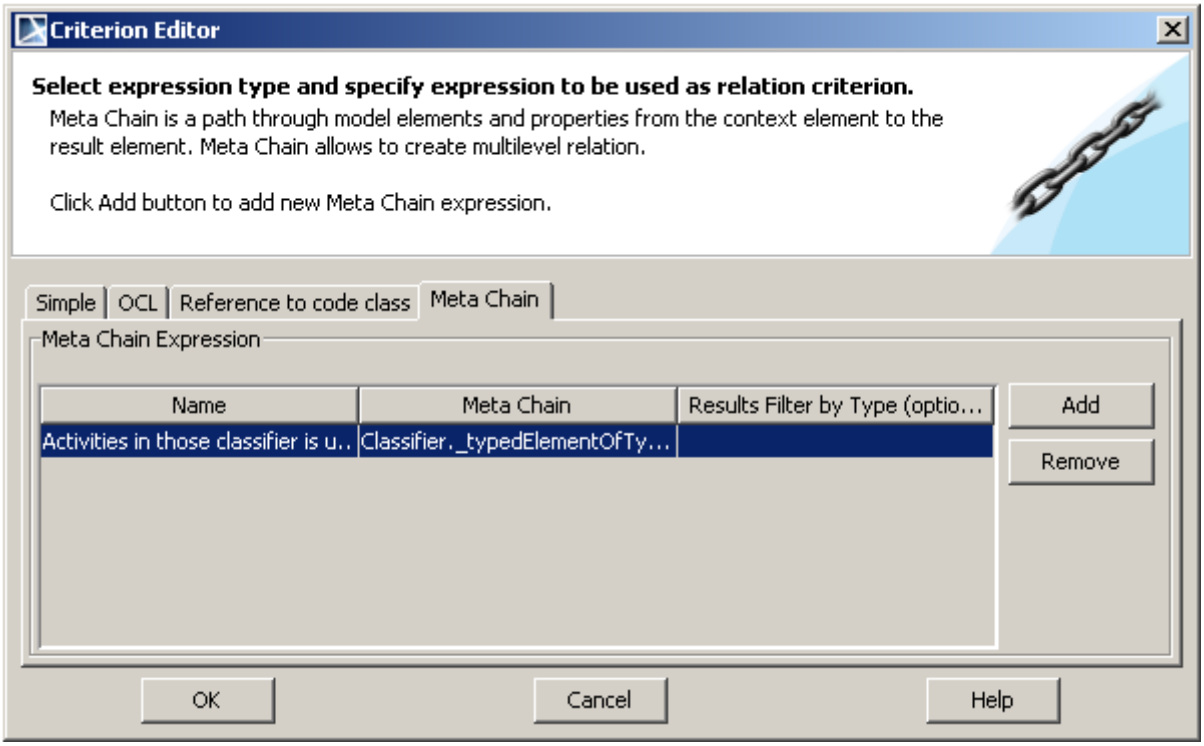


Figure 32 -- Criterion Editor dialog for defining multi properties chain expressions

Column	Description
Name	A name of a meta chain expression. It is mandatory and has a default value “Untitledn”, where “n” stands for the expression sequence number. You can edit the name either directly in a cell, or in the **Name** dialog, opened by clicking the “...” button in this cell. To create an expression, click **Add**. To remove an expression, click **Remove**.
Meta Chain	A meta chain expression, containing the so-called links, i.e., pairs of a metaclass/ stereotype and a property/ tag. You can edit the expression either directly in a cell, or in the **Meta Chain Expression** dialog (see Figure 33 on page 65), opened by clicking the “...” button in this cell.
Results Filter by Type	A type of the result elements defined in the final link of the meta chain expression. Note that only this type elements will be displayed as derived property values. Selecting more than one result type for a meta chain expression is allowed. To open the dialog for selecting the result element types, click the “...” button.

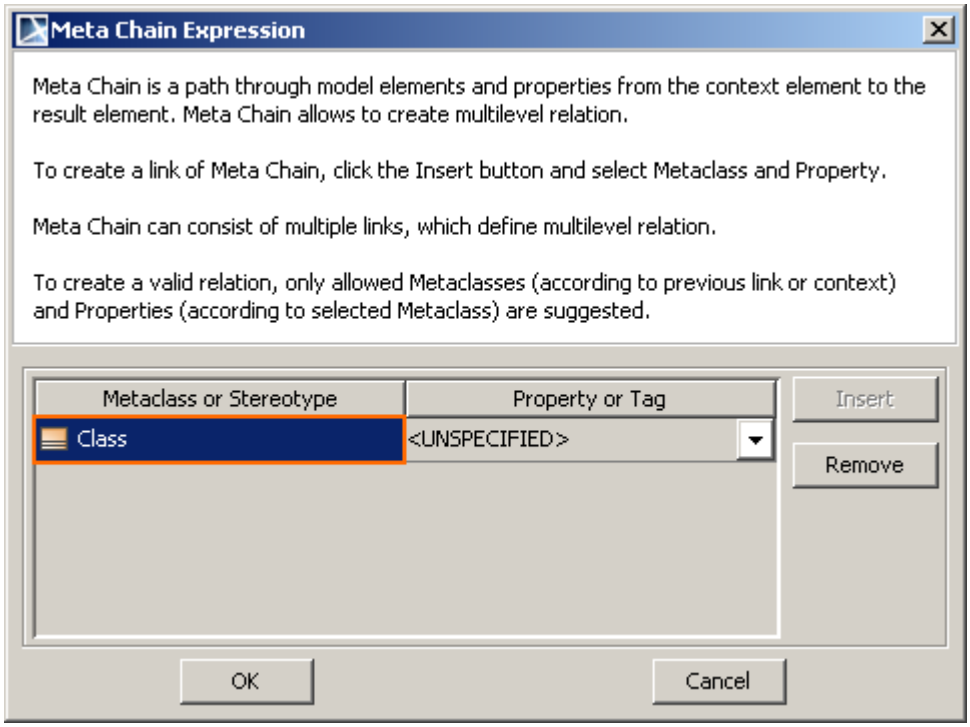


Figure 33 -- Meta Chain Expression dialog for defining links of a meta chain expression

Column	Description
Metaclass or Stereotype	A metaclass or a stereotype of a link, contained in a meta chain expression. To create a link in a meta chain expression, click **Add**. To remove a link from a meta chain expression, click **Remove**. The first link, when created, already has a metaclass or a stereotype selected by default as it is shown in Figure 33 on page 65. This value comes from the derived property specification and usually is the element, which is set as a customization target for the derived property. NOTE: If a derived property has more than one customization target, the default metaclass or stereotype of the first link is common for all the customization targets of this derived property. For example, if there are three customization targets, such as boundary, control, and entity, the first link of the meta chain expression will be a class, as it is for all of them. The default metaclass or stereotype of the first link can be changed. The second link is more specific: the values suggested for selection are limited according to the values selected in the first link, since you will not be able to add a new link until the current link is not specified. The same rule is valid for the subsequent links.
Property or Tag	A property or a tag of a link, contained in a meta chain expression. You can choose a value from the list, which contains properties (including the derived ones) and tags specified for the link's metaclass or stereotype.

Example:

Let's say we have a customization class with the class element type set as a customization target. We will create two derived properties for this customization class: one named "directlyRelatedClasses" and the other named "indirectlyRelatedClasses".

For the "directlyRelatedClasses" derived property we will specify a meta chain expression searching for the directly related classes (see Figure 34 on page 66). This expression contains two links (one step).

For the “indirectlyRelatedClasses” derived property we will specify a meta chain expression searching for the indirectly related classes (via the other classes) (see Figure 35 on page 67). This expression contains four links (two steps).

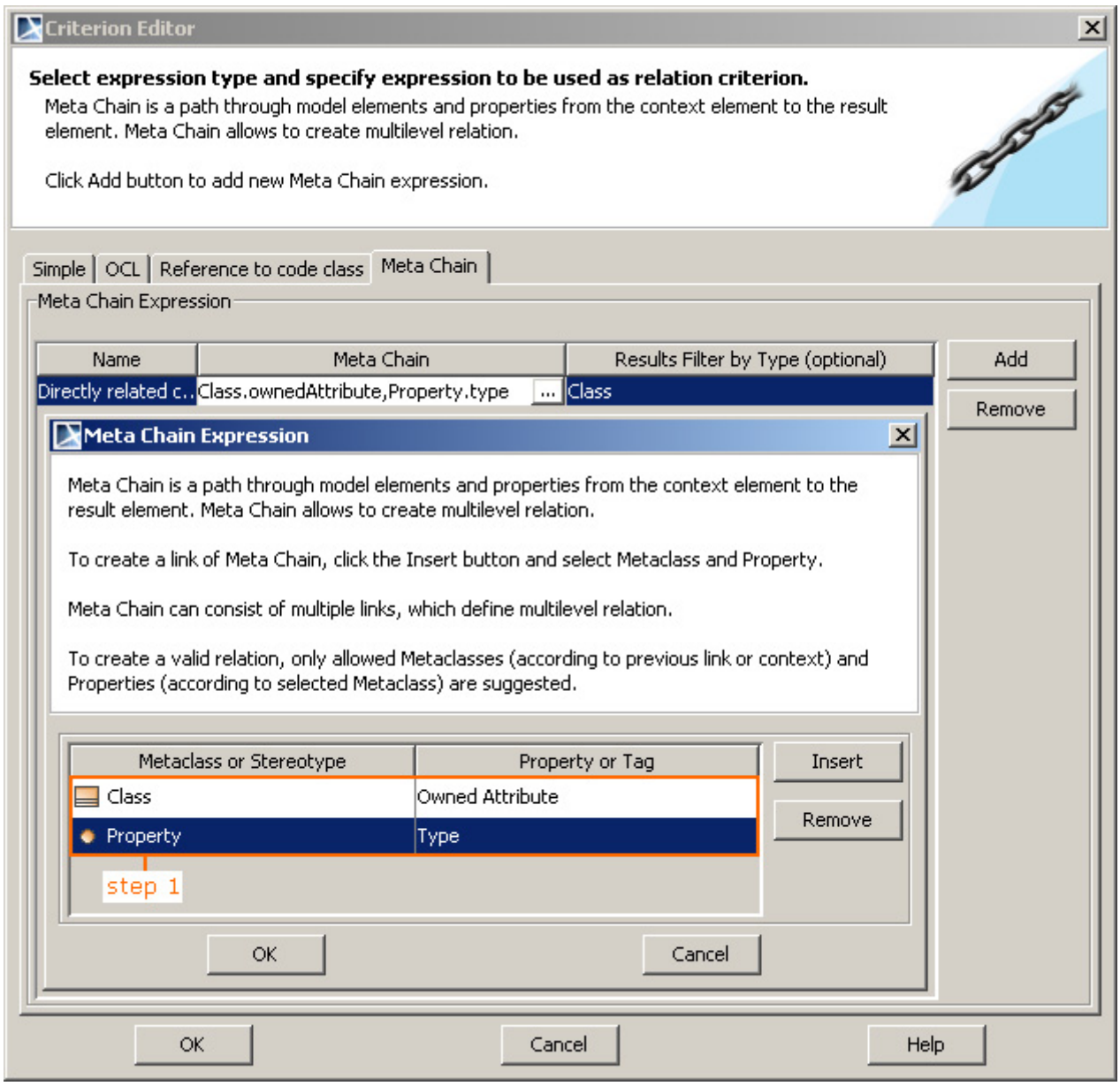


Figure 34 -- A meta chain expression for the Directly Related Classes derived property (one step)

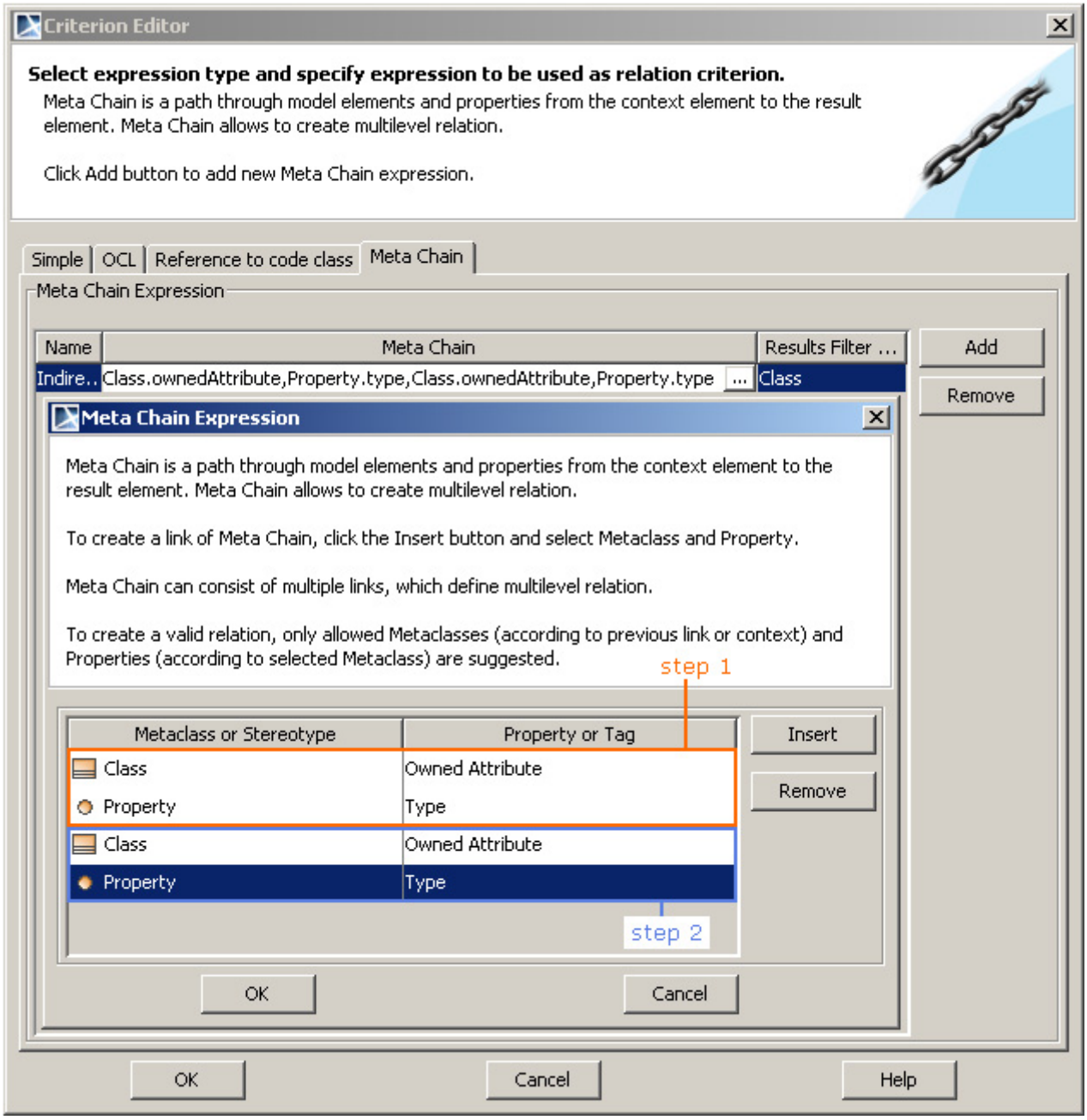


Figure 35 -- A meta chain expression for the Indirectly Related Classes derived property (two steps)

Thereafter, we will create a robustness diagram and three classes, each with a different stereotype: the “UserBrowser” boundary, the “UserManager” control, and the “User” entity, related by the associations as it is show in Figure 36 on page 68.

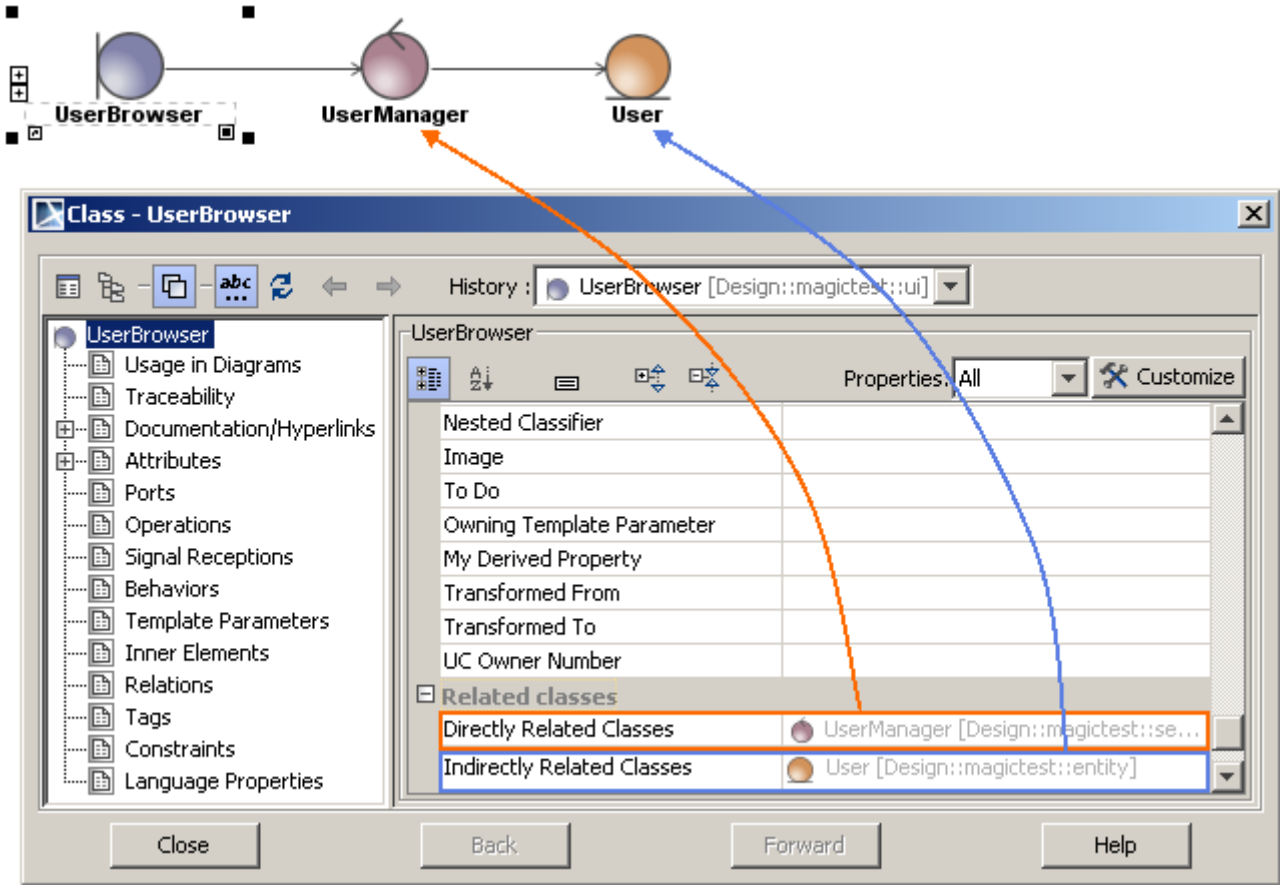


Figure 36 -- A Directly Related Classes and Indirectly Related Classes derived property values in the extended element’s Specification window

Now you can see the **Directly Related Classes** and **Indirectly Related Classes** properties in the Specification window of the “UserBrowser” boundary. The **Directly Related Classes** property value points to the “UserManager” control, and the **Indirectly Related Classes** property value points to the “User” entity.

Note that Figure 36 on page 68 shows the derived properties in the **Related classes** property group. As this example does not show how to create property groups and subgroups, and assign to them derived properties, please, refer to the following sections:

- “Creating your own property groups and subgroups” on page 24.
- “Derived properties visibility” on page 69.

Recursive properties

A derived property gathering all property values recursively of the property that is specified for the result elements.

A recursive property is defined as a binary class “com.nomagic.magicdraw.derivedproperty.RecursiveExpression()” with parameters <property-name> and <recursive-property-name>.

For an example of the recursive derived property, please, refer to the section “Binary expressions” on page 60.

With help from the recursive properties, you can collect and represent, for example:

- All the elements, which are directly or indirectly related to the specific one through the generalization relationships.

TIP!	For an example, refer to the derived property customization expression, which is available in <MagicDraw installation directory>\profiles\Traceability Customization.mdzip. Go to Traceability customization::Properties descrip-tors::Realization::Classifier Realizing Classifiers::moreSpecificClassifiers.

- All the elements, which are directly or indirectly related to the specific one through the abstraction and other types of relations.

TIP!	For an example, refer to derived property customization expression, which available in <MagicDraw installation folder>\profiles\Traceability Customization.mdzip. Go to Traceability customization::Properties descrip-tors::Realization::Element All Realizing Elements::allRealizingElements.

Expressions merge

Let's suppose we have two customization classes extending the same UML element, i.e., the same element is specified as a customization target in both customization classes. Each customization class has a derived property. If the derived properties' names, types, and multiplicities are the same, then the expressions defined for these derived properties are merged. Consequently the extended UML element will have only one derived property instead of two and the property values will be calculated according to the union of expressions defined in both derived properties.

This situation occurs when the derived property, which is already specified in, for example, the UML Standard Profile, is also specified in a project. Note that in contrast to the example above the real-life expression merge can be performed in more complex situations with more than two derived properties having more than one expression defined. The expression merge can make unions of different types of expressions.

Derived properties visibility

A newly created derived property by default is visible in the following places:

- General specification pane of the extended UML element's Specification window.
- **Properties** panel of the extended UML element.
- **Compartment Edit** dialog.

IMPORTANT!	To see derived properties of customized elements which has usedUMLProperties tag specified, make sure derived properties are assigned as values for the usedUMLProperties tag.

- **Criterion Editor** dialog for editing relation criteria in a Relation Map diagram.
- **Select Dependency Criteria** dialog for editing the dependency criteria in a Dependency Matrix diagram.

Moreover, DSL customization allows for being more precise, when specifying a derived property's visibility as this can be done through the specification of the extended element's properties group or subgroup, to which the derived property is assigned. As a result, the derived property will be visible in the group or subgroup according to this group's or subgroup's visibility settings.

To make derived properties of customized elements which has usedUMLProperties tag specified visible

1. Open the Specification window of customization element.

2. Open the **Tags** specification pane.
3. Select the **usedUMLProperties** tag and assign one or more derived properties as values to this tag.
4. Save and reload the project.

To assign the derived property to a property group

1. Select the customization class with the derived property.
2. Create a property group for the customization class (perform 1 to 4 steps of the procedure “To create a property group” on page 25).
3. In the **Tags** specification pane, edit tag values that specify the property group’s visibility.
4. In the same pane, click on the “properties” tag and then click the **Create Value** button.
5. In the **Item Filter** dialog, select the check box near the appropriate derived property and click **OK**. As it is depicted in the figure below, the selected derived property will be assigned to the “property” tag as a new value.
6. Click **Close**.

The steps for assigning the derived property to a property subgroup is a parallel case. You just have to create a property subgroup in addition and assign the derived property not to the property group, but to the subgroup.

IMPORTANT!	It is also allowed to assign a derived property to a property group, which is specified not in the same customization class, if both customization classes extend the same UML element.

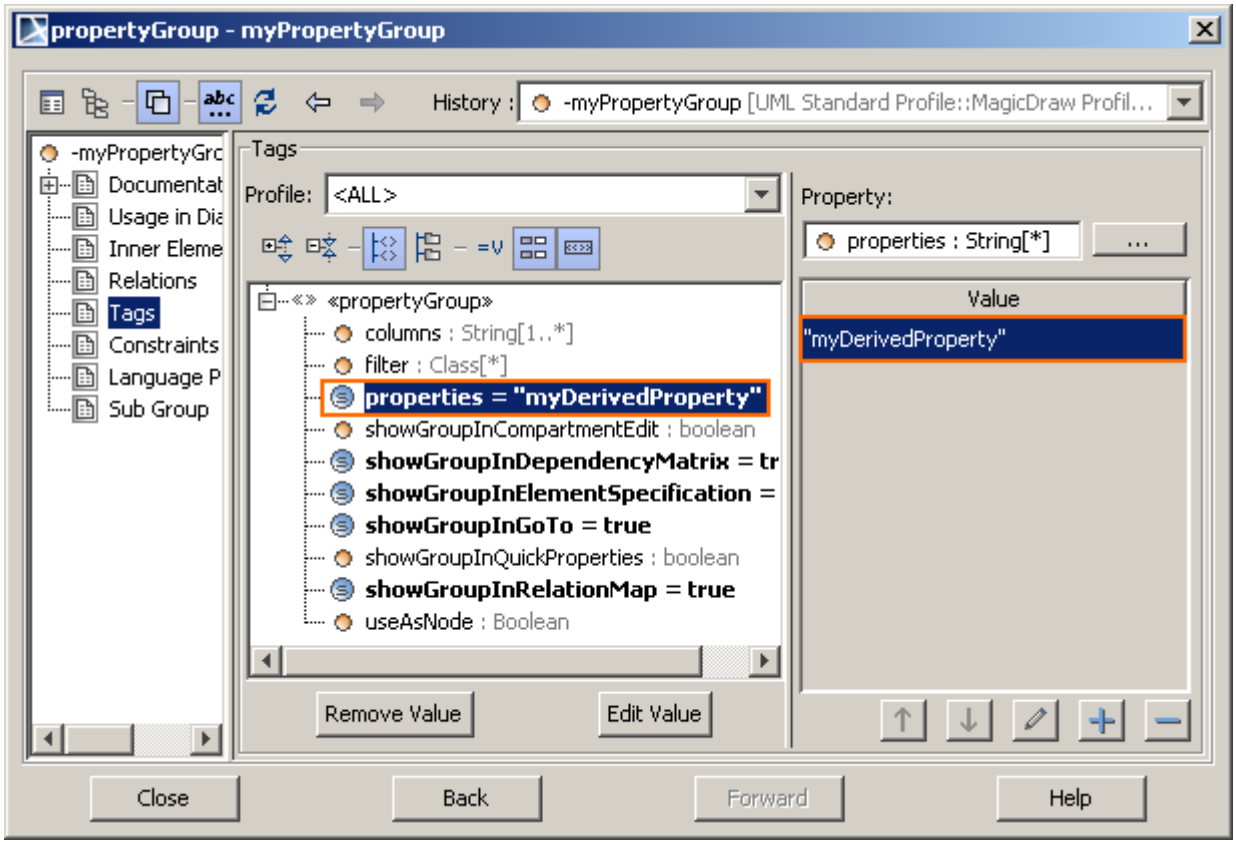


Figure 37 -- Derived property assigned to property group

For more information about creating the extended property groups and subgroups, and defining their visibility, see section “Creating your own property groups and subgroups” on page 24.

Moreover, all the derived properties, together with custom UML properties, are available for creating your own Report templates.

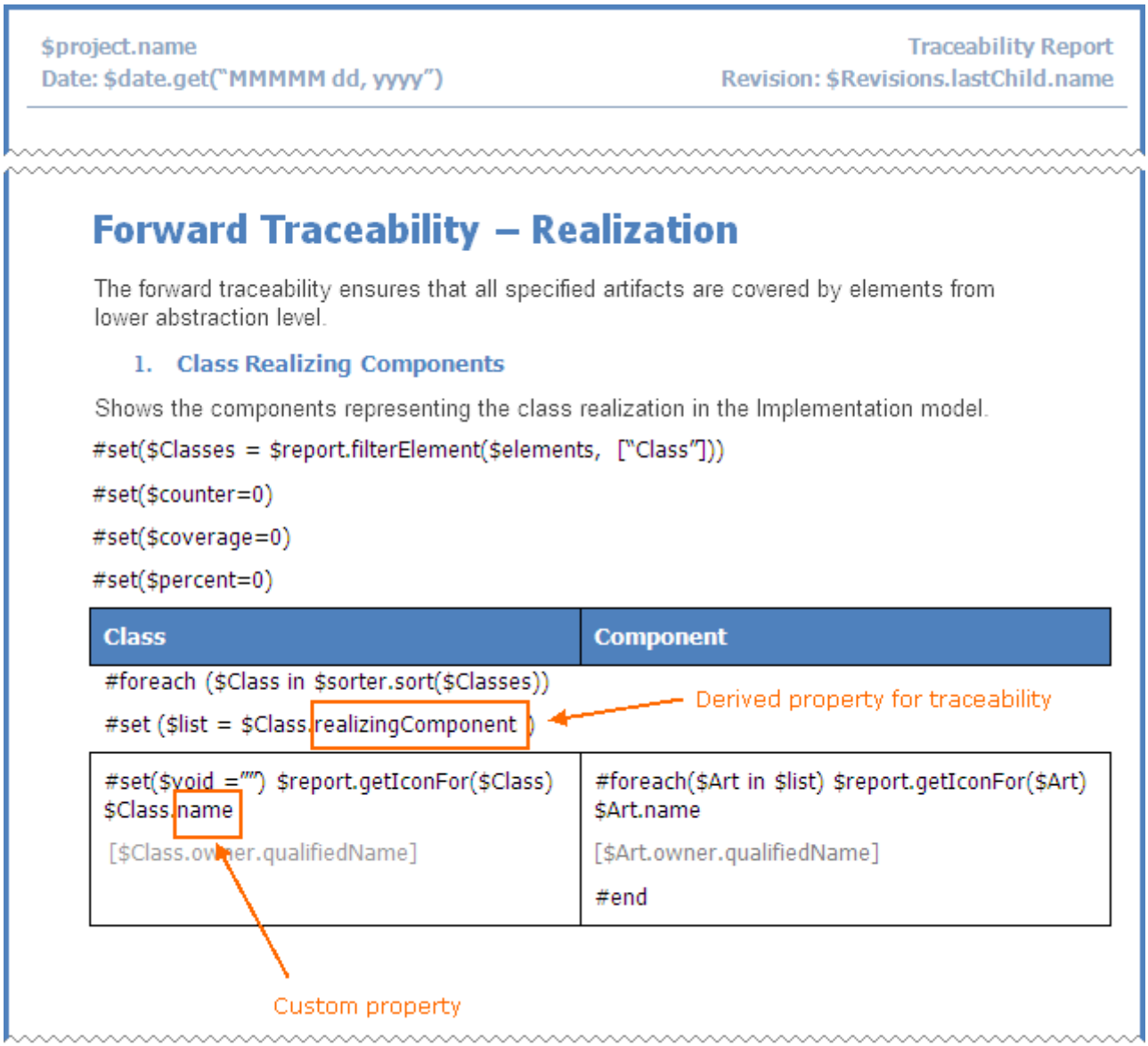


Figure 38 -- Derived properties in Traceability Report template

NEW! Creating numbering customizations

You can create numbering formats (schemes) and apply them on DSL elements (i.e., both DSL types and stereotyped elements) in MagicDraw or its add-in program, such as the SysML, Cameo Business Modeler, or UPDM plugin.

You can create one or more numbering schemes for the same DSL element. Moreover, the same numbering scheme can be applied on several DSL elements. In this case instances of different DSL elements will be numbered in sequence. For example, all actors and use cases in a use case diagram will be numbered in sequence, if the same numbering customization is used for both actors and use cases. Another example of numbering different DSL elements in sequence is shown in the following figure. As you can see, start events,

end events, tasks, and gateways in the BPMN Process diagram are numbered using the same numbering scheme.

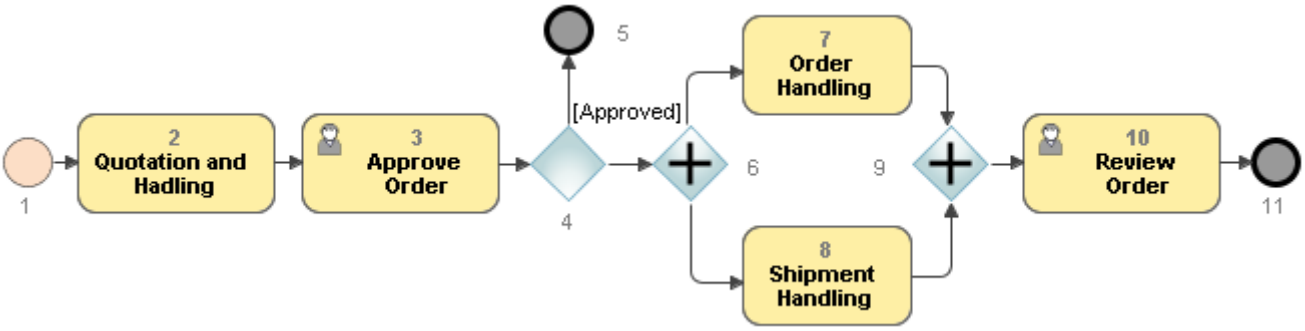


Figure 39 -- Numbering instances of different BPMN elements in sequence

In this section you will find the instructions how to customize the generic numbering mechanism in order to create your own numbering scheme. For better understanding the further material first of all read the following definitions.

DEFINITIONS

Numbering scheme	Defines a numbering style as well as number parts that are used to compose a DSL element number. A numbering schema can have one or more number parts. Numbering scheme is represented as a class with the «NumberingScheme» stereotype applied.
Number part	Represents a rule for calculating an individual part of the whole element number, as the element number is composed of one or more individual number parts. It can be a number, character, separator, or other. Number part is represented as a numbering scheme property with the «NumberPart» stereotype applied.
Numbering property	Indicates a DSL element property wherein the element number will be stored and defines a numbering scheme that will be used for the DSL element numbering. Numbering property is represented as a customization class property with the «AutoNumber» stereotype applied.

Learn more about creating numbering customizations in the following sections:

- "Basic steps for creating numbering customization" on page 72.
- "Creating your first numbering customization" on page 73.
- "Relevant tag values" on page 80.

Basic steps for creating numbering customization

In order to create a numbering customization for an element, you have to perform the following steps:

1. Create a stereotype to customize a desired element.
2. Add a new property to the stereotype.

IMPORTANT! This property will be used to store the element number.

3. Create a numbering scheme to define a numbering style and number parts.
4. Create a customization class for the previously created stereotype.
5. Add a numbering property to the customization class.
6. Reload the project to apply changes.

Creating your first numbering customization

Let's assume that we need to number packages as shown in the following figure.

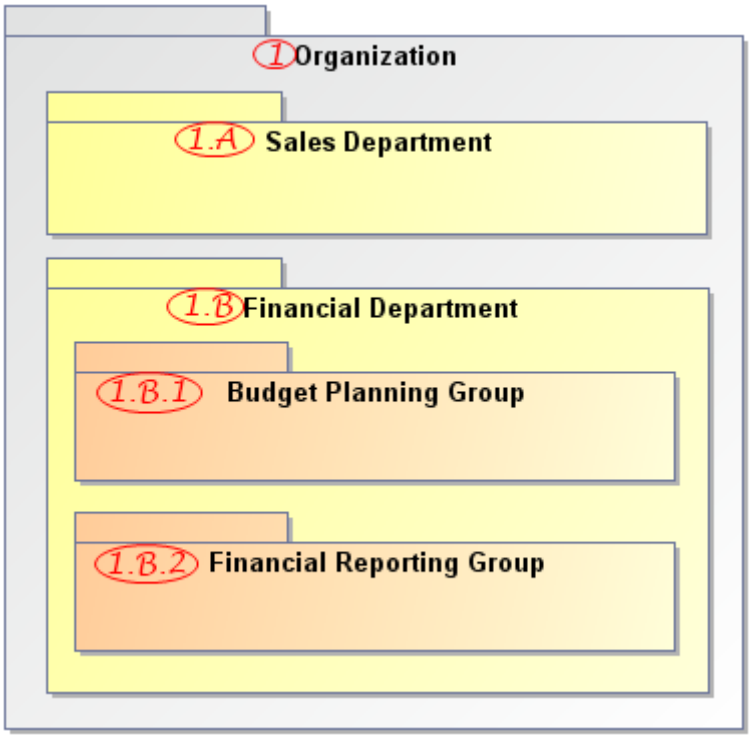


Figure 40 -- Task for numbering packages

There should be three numbering levels. Rules for calculating package numbers in each level are described in the following table.

Numbering level	Numbering format	Examples
1st	<number>	"1", "2", "3"
2nd	<number><separator><character>	"1.A", "1.B", "2.A", "2.B"
3rd	<number><separator><character><separator><number>	"1.A.1", "1.A.2", "1.A.3", "2.B.1", "2.B.2"

So we need to create a numbering scheme and use it in the numbering customization for package elements. For this purpose we will perform the steps described in "Basic steps for creating numbering customization" on page 72 one by one.

NOTE We will perform this task in the Quick Start perspective.

1. Creating a stereotype to customize a desired element

To create a stereotype that customizes UML packages

NOTE We will use a profile diagram to create the stereotype. You are free to choose any other way you know for creating stereotypes.

1. Create a profile diagram.
2. On the diagram pallet, click the **Stereotype** button.
3. Click a free space on the diagram pane. A shape of the new stereotype will appear, and the **Select Metaclass** dialog will open.
4. Select the Package metaclass and click **OK**.
5. Name the stereotype “Numbered Package”.

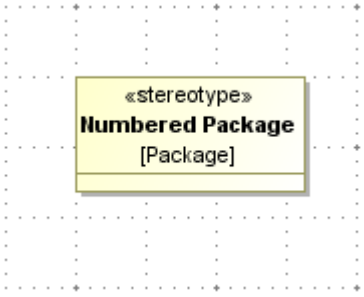


Figure 41 -- Stereotype for customizing UML packages

TIP! You can make the «Numbered Package» stereotype invisible, if you want the stereotype and its properties to be hidden on a diagram. For more information about invisible stereotypes refer to "Invisible stereotypes, tags and constraints" on page 11.

2. Adding a new property for storing numbers

To add a new property for storing package numbers to the “Numbered Package” stereotype

1. Select the shape of the “Numbered Package” stereotype on the profile diagram pane.
2. Add a new stereotype property named “No.”.

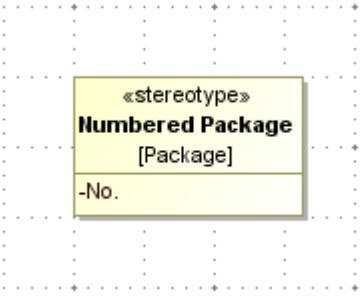


Figure 42 -- Property for storing package numbers

3. Creating a numbering scheme to define a numbering style and number parts

To create a numbering scheme

NOTE We will use a profile diagram to create the numbering scheme. You are free to choose any other way you know for creating classes with stereotypes applied.

1. On the diagram pallet, click the **Class** button.
2. Click a free space on the diagram pane. A shape of the new class will appear.
3. Name the class “Package Numbering Scheme”.

   NOTE The name of the numbering scheme will be displayed in the **Element Numbering** dialog. It is recommended to give it a short and meaningful name.

4. Apply the «NumberingScheme» stereotype on the class.

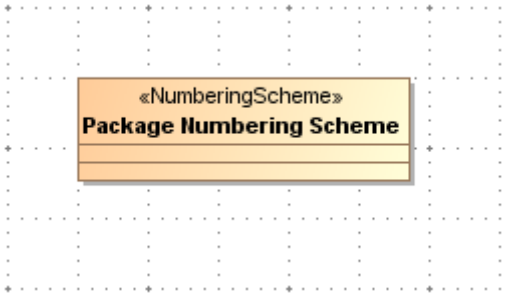


Figure 43 -- Scheme for numbering UML packages

To define a numbering style

1. Open the Specification window of the just created numbering scheme.
2. Open the **Tags** specification pane.
3. Double-click the **numberingStyle** tag. The tag value will be set to **Multi-level**, the default value. Leave the value as it is, because we need to have three numbering levels for numbering packages.

   NOTE For detailed descriptions of the **numberingStyle** tag values refer to "numberingStyle tag values" on page 80.

4. Click **Close**, when you are done.

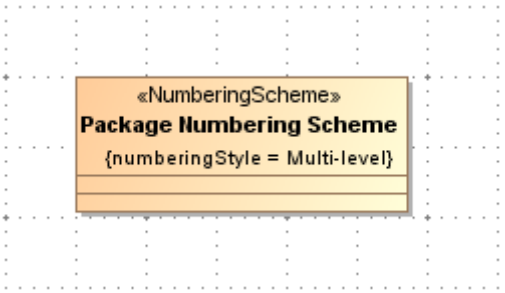


Figure 44 -- Numbering scheme with multilevel numbering style defined

The following table shows number parts that should be created to compose numbering formats described in the preceding table.

Number part	Number part name
<number>	Level 1
<separator>	Dot 1
<character>	Level 2
<separator>	Dot 2
<number>	Level 3

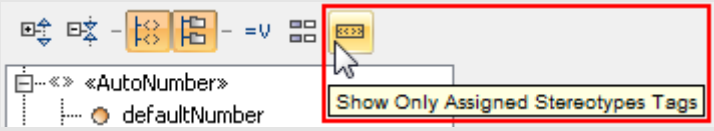
To create a number part

1. Select the shape of the numbering scheme on the profile diagram pane.
2. Add a new numbering scheme property.
3. Apply the «NumberPart» stereotype on the property.

To specify a number part for the first numbering level

1. Create a number part named “Level 1” (see the procedure "To create a number part" on page 76).
2. Open the **Tags** specification pane in the Specification window of the number part.

TIP! Click Show Only Assigned Stereotypes Tags on the pane toolbar to see only those tags that belong to the «NumberPart» stereotype.



3. Double-click the **sequence** tag. The tag value will be set to **Numeric**, the default value. Leave the value as it is, because we need to have numbers at the first numbering level.

NOTE For detailed descriptions of the **sequence** tag values refer to "sequence tag values" on page 80.

4. Double-click the **initialValue** tag and define “1” as its value, because the numbering at the first level should start from 1.
5. Click **Close**, when you are done.

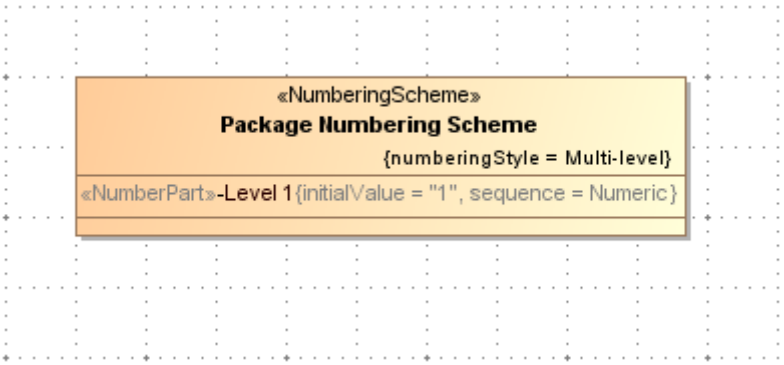


Figure 45 -- Numbering scheme with number part defined for first numbering level

To specify a number part for the first separator

1. Create a number part named “Dot 1” (see the procedure "To create a number part" on page 76).
2. Open the **Tags** specification pane in the Specification window of the number part.
3. Double-click the **sequence** tag.
4. In the **Value** drop-down list, select **Separator** to change the default tag value.
5. Double-click the **initialValue** tag and define “.” as its value, because the separator between the first and the second numbering level should be a dot.
6. Click **Close**, when you are done.

To specify a number part for the second numbering level

1. Create a number part named “Level 2” (see the procedure "To create a number part" on page 76).
2. Open the **Tags** specification pane in the Specification window of the number part.
3. Double-click the **sequence** tag.
4. In the **Value** drop-down list, select **Character** to change the default tag value, because we need to have letters at the second numbering level.
5. Double-click the **initialValue** tag and define “A” as its value, because the numbering at the second level should start from A.
6. Click **Close**, when you are done.

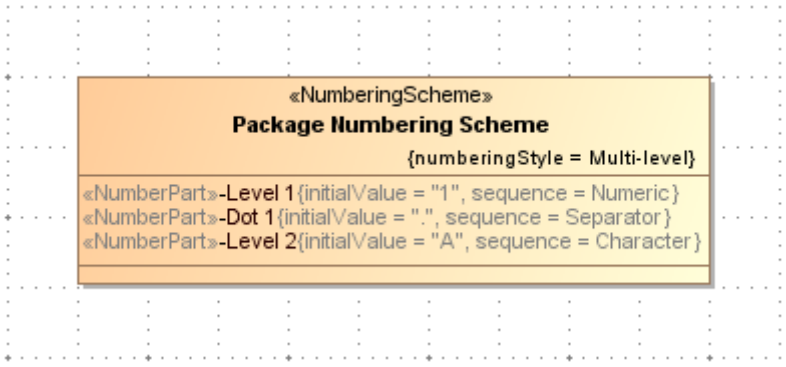


Figure 46 -- Numbering scheme with number parts defined for second numbering level

The rest of number parts are created appropriately.

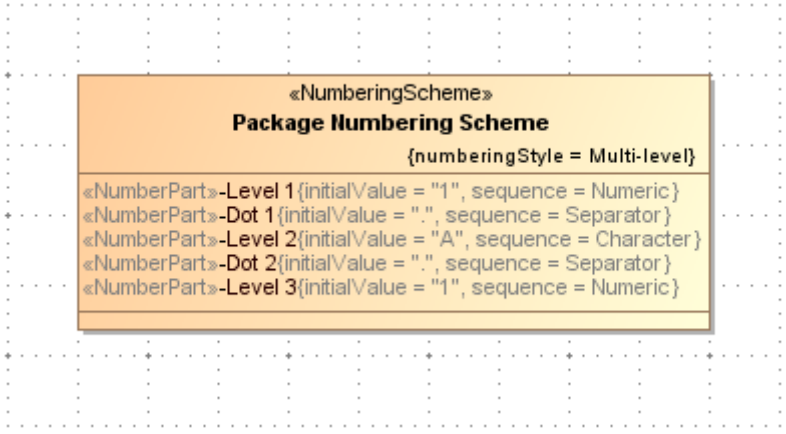


Figure 47 -- Numbering scheme with all number parts defined

NOTE As there are no number parts specified for the fourth and further numbering levels, appropriate packages will be numbered using number parts of the third numbering level. For example, “1.A.1.1”, “1.B.1.2”, and so forth.

4. Creating a customization class for the stereotype that customizes the desired element

In order to use the just created scheme for numbering packages, we need to define it in the numbering customization for package elements. For this a customization class must be first created.

To create a customization class for the “Numbered Package” stereotype

NOTE	We will use a profile diagram to create the customization class. You are free to choose any other way you know for creating customization classes.

1. On the diagram pallet, click the **Class** button.
2. Click a free space on the diagram pane. A shape of the new class will appear.
3. Name the class “Numbered Package Customization”.
4. Apply the «Customization» stereotype on the class.
5. Open the **Tags** specification pane in the Specification window of the customization class.
6. Double-click the **customizationTarget** tag, select the “Numbered Package” stereotype in the element Selection dialog, and click **OK**.
7. Double-click the **showPropertiesWhenNotApplied** tag. The tag value will be set to *true* meaning that all properties of the “Numbered Package” stereotype will be visible as properties of the Package metaclass even if the stereotype is not yet applied.

TIP!	For more about the **showPropertiesWhenNotApplied** tag, refer to "Always visible properties" on page 30.

8. Click **Close**, when you are done.

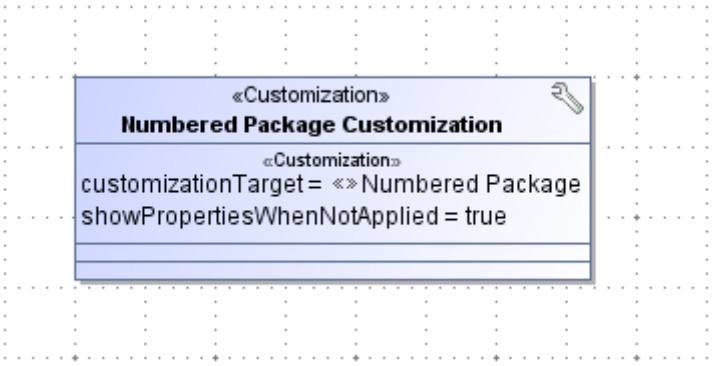


Figure 48 -- Customization class for “Numbered Package” stereotype

5. Add a numbering property to the customization class

After the customization class for the “Numbered Package” stereotype has been created, we can specify the stereotype property for storing element numbers as well as define a numbering scheme that will be used. For this a numbering property must be created.

To add a numbering property to the customization class of the “Numbered Package” stereotype

1. Select the shape of the customization class.
2. Add a new customization class property named “Package Numbering”.
3. Apply the «AutoNumber» stereotype on the property.
4. Open the **Tags** specification pane in the Specification window of the property.
5. Double-click the **numberedProperty** tag, select the “No.” property of the “Numbered Package” stereotype in the element Selection dialog, and click **OK**.
6. Double-click the **numberingScheme** tag, select the “Package Numbering Scheme” class in the element Selection dialog, and click **OK**.

7. Change the **defaultNumber** tag value to *true*. The *true* value means that this numbering property will be set as default, when numbering instances of the customized Package metaclass.
8. Click **Close**, when you are done.

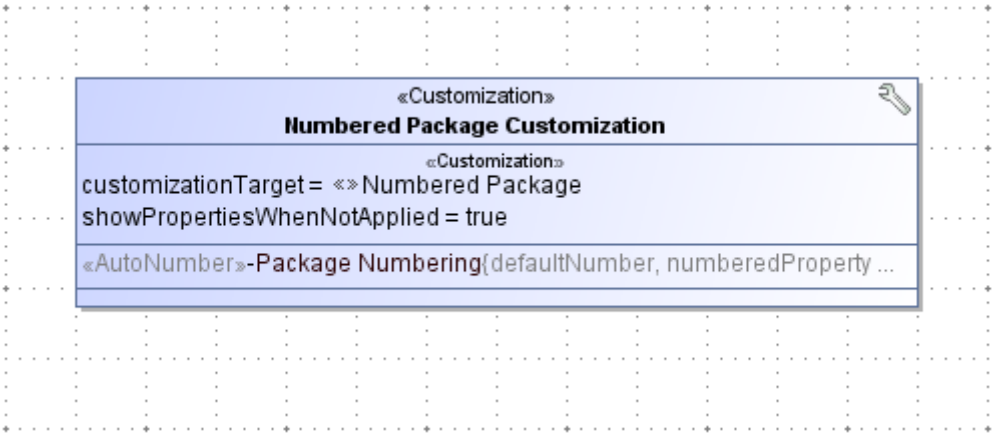


Figure 49 -- Numbering property in customization class of the "Numbered Package" stereotype

6. Reload the project to apply changes

Open the appropriate package diagram and see the packages numbered using the just created numbering customization.

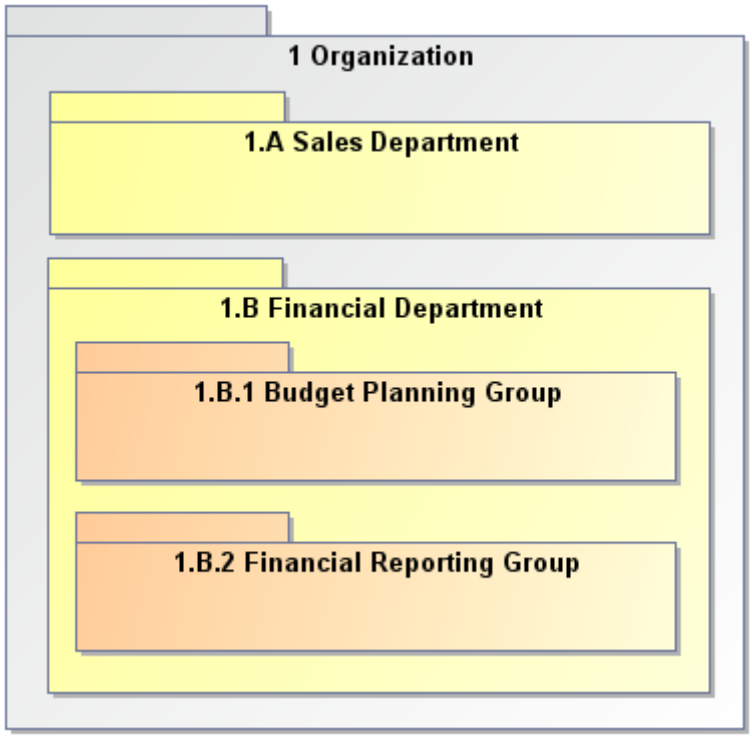


Figure 50 -- Packages numbered using numbering customization

Relevant tag values

numberingStyle tag values

Tag value	Description
Multi-level	Select this style, if you need to number elements in several numbering levels. Specify at least one number part for each numbering level. Top level elements will be numbered using the first number part. Elements in the subsequent levels (i.e., those that are directly or indirectly contained in the top level element) will be numbered using successive number parts. **NOTE:** If you have specified less number parts than there are element levels, the last element number part of the separator type and the last element number part of the non-separator type (for example, numeric or character) will be reused to number a subsequent level elements. For instance, if number parts are specified only for two numbering levels, and elements in the second level are numbered as “1.A”, “1.B”, and “1.C”, elements in the third numbering level will be numbered as “1.A.A”, “1.A.B”, and so on.
Consecutive	Select this style, if you need to number elements in one numbering level. Specify only one number part. **NOTE:** If you have specified more than one number part, elements will be numbered using only the last number part. All other number parts in this case will be treated as static. For instance, if an element number includes five number parts (for example, “1.1.1”). and the consecutive numbering style is selected, then all elements will be numbered by changing only the last number part, i.e., “1.1.**1**”, “1.1.**2**”, “1.1.**3**”, and so forth. The same happens after changing multi-level numbering style to consecutive.

sequence tag values

Tag value	Description
Numeric	Select this type, if you need to use only positive numbers for calculating a value of the number part. **NOTE:** Elements will be numbered starting with “1” in ascending order, if no initial value is specified.
Character	Select this type, if you need to use only Latin letters for calculating a value of the number part. **NOTE:** Elements will be numbered starting with “A” in ascending order, if no initial value is specified.
Expression	Select this type, if you need to use an OCL or Binary (Reference to code class) expression for calculating a value of the number part. Use the **Expression Editor** dialog to define the expression.
Separator	Select this type, if you need to specify a separator between two number parts. **IMPORTANT!** Do not forget to specify the separator value in the **initialValue** tag. If the tag value is not specified, the separator will not be used.
OwnerNumber	Select this type, if you need to use the number of an owner that is numbered using another numbering customization for calculating a value of the number part.